Metrizable symbolic data structure for ill-defined problem solving

Axel Palaude [a], Chloé Mercier [a,*] and Thierry Viéville [a,**]
[a] *Mnemosyne Team, Inria Bordeaux, U. Bordeaux, LaBRI and IMN, France*
E-mails: axel.palaude@inria.fr, chloe.mercier@inria.fr, thierry.vieville@inria.fr

Submitted to Neurosymbolic Artificial Intelligence

Abstract. Within the context of cognitive computational neuroscience modeling and neurosymbolic artificial intelligence, using both symbolic and numerical approaches, we propose a method for relating explicit, readable data structures to biologically plausible numerical operations. We consider a parameterized edit distance between two hierarchical symbolic data instances, thereby embedding the data in a metric space and enabling the definition of a geodesic between two data points. We also define in this symbolic space the notion of data region, here equivalent to data type or concept, and provide a projection onto such a region, so that very generic machine learning algorithms can now be applied simultaneously at both the symbolic and numerical levels. This theoretical work may be of interest to researchers in computational neuroscience, model simulation, and artificial intelligence who aim to design explainable, explicit, ill-defined problem-solving mechanisms. The advantage is that it provides a straightforward data representation on which generic numerical planning or learning algorithms can be applied directly, once the data-structure distances are adequately parameterized for the given application domain. The proposed specification is not only formally developed but also implemented in detail at the programming level, with experimental illustrations that demonstrate feasibility and enable evaluation and use of the proposed approach.

Keywords: Symbolic specification, Edit distance, Problem solving, Metric space

1. Introduction

Within the context of cognitive computational neuroscience modeling and neurosymbolic artificial intelligence, using both symbolic and numerical approaches, we propose a method for relating explicit, readable data structures to biologically plausible numerical operations. To this end, a set of symbolic data structures is equipped with an edit distance, yielding a functional metric space on which powerful numerical algorithms can be applied directly.

What is a symbol? The vanilla definition is that a symbol is a mark or label that indicates or is considered as representing a subject or object, its properties, or relationships between them. Symbols are also used to go beyond what is known or perceived by creating linkages between concepts or experiences.

This notion is the foundation of human knowledge. At first glance, at the syntactic level, a symbol is an "atom of knowledge", and is no more than the label (or identifier) of an object in the broad sense. Furthermore, it has a "meaning" when it is semantically *grounded*, in the sense discussed by [35]. The link between such a label and the object to which it refers has been examined and debated in [62], which situates this notion within more

*Supported by https://team.inria.fr/mnemosyne/en/aide.E-mails: axel.palaude@inria.fr, chloe.mercier@inria.fr.
**Corresponding author. E-mail: thierry.vieville@inria.fr.

recent cognitive studies. This is the "symbol grounding problem", understood as the process of embedding symbolic computations onto sensorimotor features of real-life objects and behavior. This provides a semantic interpretation of the symbolic system, or model (in the sense of a model as a set of logical assertions), which involves the capacity to select referents for concepts and yields the notion of consciousness, as discussed in [6]. As noted by these three references, this includes affordance, which is not only a real object feature but also the capability to interact with it to attain an objective and receive an outcome.

At the Artificial Intelligence (AI) level, unless embodied in a robot with as much experience as our brain, which is through the phylogenetic evolution in interaction with the external environment for millions of years, symbols are not grounded in the real world, even if they adequately model some aspects of reality. Furthermore, if only prediction or generative performances are targeted regarding an AI system that processes symbols such as words, as in LLM frameworks, there is no need to manually define a link between symbolic input and output and numeric representations in the underlying artificial neural networks and related mechanisms: big data machine learning mechanisms make the job. However, as reviewed recently by [28] regarding neurosymbolic mechanisms, it could also be necessary to construct the relation between a symbolic data structure and its numeric anchoring explicitly. This occurs for explainable AI [16]. This also occurs when a priori explicit knowledge is required to constrain very general AI mechanisms to a specific field of inference and to control and avoid biases. Moreover, for parsimonious AI, it is interesting to introduce as much well-known a priori knowledge as possible rather than estimating it from large datasets.

The main issue addressed here is thus to propose an explicit, parameterizable, therefore adaptable, link between symbolic representation of knowledge and its numerical anchoring to be either used within usual AI numerical algorithms or as a way to model cognitive representations.

Organization of the paper. To explain the pertinence and originality of our proposal, we situate it in the next section in relation to existing neurosymbolic approaches. Then, the problem-solving issue is taken as a significant example. We also situate this work on brain modeling at the symbolic level, both with respect to the paradigm and to knowledge representation.

Given these prerequisites, we detail a modeling approach that instantiates the previously stated general principles, thereby motivating the specification developed in this paper. We describe, at the computational level, how the previous data structure ingredients can be formalized, including at the programming level, thereby allowing us to consider generic numeric algorithms directly on the symbolic data structures. This includes embedding symbolic data structures in a metric space and enjoying a notion of geodesics. This also includes defining a concept corresponding to a region of the state space, equipped with a projector that maps a data point to that region. We also introduce the notion of a bound in this metric space to enable exploration beyond the given data structures.

At a further step, the development of a metrizable symbolic data structure embedded in a metric space for use in general machine learning mechanisms also requires specifying a scalar field, along with interpolation and barycenter mechanisms. All of this is then illustrated by presenting a few toy demonstrations and by reporting a realistic experiment.

As a final step, we discuss how generic machine learning algorithms can be applied within this formalization, particularly for open-ended or ill-defined complex problem-solving. We finally discuss the perspectives of this work before a brief conclusion.

2. Context and related work

2.1. Neurosymbolic approaches

The numerical machine learning approach is primarily based on statistical inference, including Bayesian methods, and has achieved outstanding performance. On the other hand, when manipulating knowledge, an explicit symbolic machine learning mechanism, thus reasoning, is much more efficient to infer knowledge from known facts, and hybrid mechanisms seem mandatory for complex tasks where non-trivial a priori knowledge is to be introduced. It also has significant advantages when interpretability (by experts) and explainability (by end users) are key features, as discussed in [16], and it develops these dual aspects by proposing criteria for evaluating both. These features

are essential if such learning mechanisms are used to model cognitive functions. In addition, formal reasoning is less computationally expensive than billions of operations in deep network computation, as studied by [22], who compare the computational costs of the two approaches, focusing on academic use by typical research teams. They also raise ecological and ethical issues that pose a significant challenge in the context of the artificial intelligence boom.

Coupling both approaches should bring the required computational power to solve complex problems or model complex brain functions, in an explainable way, and with parsimonious resource consumption, and neural-symbolic computing brings together robust learning in neural networks with reasoning and explainability via symbolic representations as recently reviewed in details in [28], who also introduced a taxonomy of numeric and symbolic coupling in algorithms. The present work aims to examine the coupling levels at which symbolic knowledge and rules are compiled in distributed computation, either via local symbol mapping or through more complex embeddings of symbolic rules; in both cases, symbolic knowledge is translated into the network architecture and parameters.

One motivation for such an approach is thus explainability, as reviewed by [14] in the domain of supervised learning, with both the capability to interpret both results and reasoning path, but also, as mentioned by the authors, to be able to introduce high-level explicit a priory knowledge which allows to better constraint the system, thus provide more efficient results, taking the “free-lunch” principle into account: the more general the algorithm, the less efficient on a specific problem. This may be another interest of the present approach: to introduce symbolic a priori information into numerical computations.

The interest in working at both the numeric and symbolic levels, consistent with the problem-solving focus here, is well illustrated by [65], which discusses a logic-based explanation mechanism for planning, dedicated to controlled hybrid systems in human interaction. The authors’ efficient approach places the symbolic layer “outside” the corresponding numeric representation. In contrast, at a more integrated level, such a plan is represented by the notion of an abstract path, as developed in this paper. This work is also in line with ontology-related reasoning, as, for instance, [36] shows how symbols can be optimally encoded in the network’s numeric variables before applying numerical-level reasoning mechanisms. In contrast, we present a dual approach: symbolic data are equipped with a metric, enabling direct application of numerical tools to them.

A step ahead, computational neuroscience models are primarily based on numerical mechanisms, except for the Vector Symbolic Architecture (VSA) [25], which is grounded in the Semantic Pointer Architecture (SPA). This approach introduces an intermediate algebraic abstraction of spiking neural network architectures, as in [17], to develop biologically plausible cognitive functionalities and human knowledge representations. This also recently includes a high-level symbolic representation, enabling reasoning in the preliminary work of [44] regarding not only deductive but also inductive and abductive reasoning, as developed in [39]. Using Vector Symbolic Architecture, implemented at the neural spiking assembly level via the Neural Engineering Framework [25], cognitive symbolic data structures can be realized as a biologically plausible memories [39].

As a complementary approach to these previous works, we are not going to study how to embed a symbolic representation onto a given numeric space, but how to equip the symbolic data structure directly with the numeric ingredients needed to apply numeric algorithms, and show how general algorithms can be used directly on the symbolic data structure by this means. To this end, in the next section, let us precisely state what a “symbolic representation" is, to make explicit our design choices to represent cognitive symbolic information. The targeted approach applies to both the system state and its related metadata, at the neurosymbolic level.

2.2. Problem-solving and planning problems

A typical class of problems discussed above concerns planning. Since [46] work, problem-solving and planning have been formalized as a trajectory problem from an initial state to a goal state, in a state space. This means avoiding “obstacle”, in other words, being compliant with respect to trajectory constraints, while reducing the distance to the goal. This problem statement has recently been generalized by [3] to complex open problems with multiple possible goals, while the state space is only partially known.

Solutions often use a distance, such as an edit distance, between two state-space points, for instance, in heuristics that generate step-by-step plans. In an early work [29], obstacle avoidance is expressed in terms of distances between potentially colliding parts and the goal (s), leading to the formulation of path planning problems as an optimal

control problem with variational-numerical solutions. In [66], generic path planning problems have been related to a parametric harmonic potential, which is tractable in high dimensions, reducing path planning to this potential optimization while yielding a biologically plausible solution to hippocampal navigation, in either concrete locations or abstract state space. In brief, the potential is repulsive around obstacles and attractive toward goals. Using a harmonic potential avoids local minima, whereas using a parametric potential avoids the curse of dimensionality, a standard limitation of potential methods: the need to sample the vast state space. As noted in [29], and as shown in [11], such harmonic control is a particular case of optimal control. A relatively efficient algorithm has been proposed in [66], which avoids the curse of dimensionality by designing a parametric potential whose parameters increase with trajectory length rather than with state-space dimension.

With respect to such approaches, the present development will provide a framework for directly considering this mechanism using symbolic data, without the non-negligible overhead of embedding and the somewhat arbitrary choice of a sampling method.

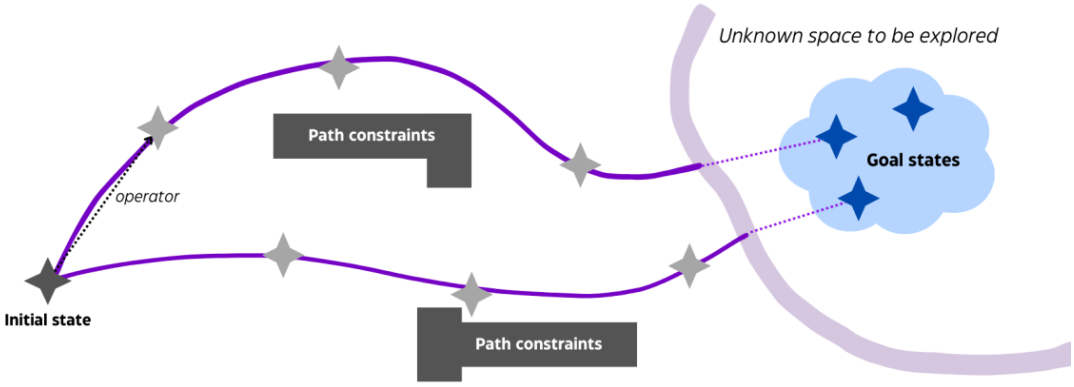


Fig. 1. Problem-solving as trajectory generation, from [39].

2.3. *Brain modeling at a symbolic level*

Studying the brain is conducted at different temporal and spatial scales, at different topological network scales, as reviewed in [8], and at different "modeling" scales, in the sense of Marr, as reviewed and questioned in [34]. At the difference of Marr's original three implementations, algorithmic and computational levels, modern vision, which is presented for instance in [31] in a comprehensive review, considers (i) the implementation level, more precisely a biophysical representation in the broad sense, and considers (iii) as the highest level the cognitive behavior, while (ii) the computational middle layer includes algorithmic aspects. This also clarifies that the numerical aspects pertain primarily to the biophysical side, whereas the symbolic representation relates mainly to the behavioral stage. This is not exclusive, but more a progression. The key point is that computational representation must intrinsically be able to manage both representations conjointly, as illustrated in Figure 2. This corresponds to the development proposed here.

Usually, the symbolic aspects of brain activity modeling are discussed at a phenomenological level, in a human-readable form. The formalization is often represented as block diagrams that show the relationships among model entities, with each entity defined by a keyword, a human-readable comment, properties, and relationships with other entities. This is essentially an informal ontology. The next step of formalization is thus to apply an effective ontology modeling.

In this direction, the brain processes can also be modeled using such a well-established framework, as explained in [30]. Brain anatomy has been formalized for a long time, as reviewed in [23]. At the same time, macroscopic cognitive models correspond to work in progress: problem-solving tasks have been addressed by [26], and focusing

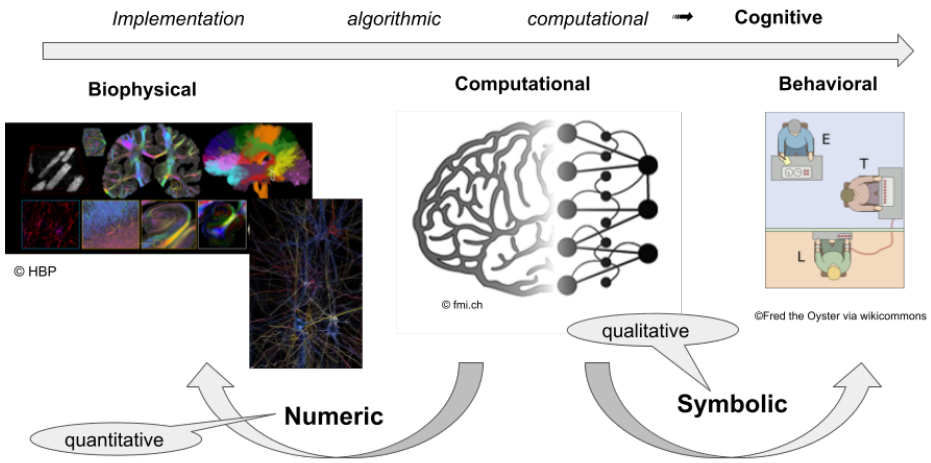


Fig. 2. A revised view of the Marr paradigm, considering modern development in computational neuroscience.

recently on computational thinking, [43] made a preliminary work on the subject, considering a subject engaged in a creative open problem-solving activity as reported in [42].

This includes both quantitative and qualitative data. Please refer to [15] for a general introduction on ontology and to [47] for a practical approach, at the modeling level. Modeling the brain explicitly using symbolic representations contrasts with the use of very general deep-learning "black-box" tools, which is really questionable, as argued by [57], raising a "free-lunch" principle at the modeling level. What predicts everything, predicts anything. The targeted approach thus addresses both the information encoded in neuronal ensembles and the modeling knowledge, thereby enabling us to explain and interpret it.

Notations and Layout. We write vectors and matrices in bold letters (bold capital letters for matrices), and scalars in italic. We use the Kronecker notation $\delta_{\mathcal{P}} \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \mathcal{P} \text{ is true} \\ 0 & \text{if } \mathcal{P} \text{ is false} \end{cases}$. We make the distinction between $\stackrel{\text{def}}{=}$ and $=$ the former being a definition, the latter the result of a derivation.

Notices:

- To make the paper easily readable, we provide verbal evidence in the text. At the same time, demonstrations of statements are given in footnotes with a title, and only straightforward derivations are required here.
- We also provide (which is uncommon in scientific papers) a large set of links to help readers from different domains understand unfamiliar terms, primarily via Wikipedia. This is only for reading help.

3. Symbolic data specification

3.1. A "natural" usual way to represent cognitive knowledge

Given the previous considerations regarding the context and motivation of this symbolic approach, we now present the types of representations targeted here, following [39]. The aim is to manipulate the internal symbolic representation of knowledge of the form shown in Figure 3, as introduced in [39]. Concepts are represented as a hierarchical data structure, in the sense of [24], which distinguishes among associative, sequential, and hierarchical cognitive memories. In our context, associative and sequential memorization structures are particular cases of hierarchical data structures: they correspond, respectively, to "map" (also called "dictionary") and "list" at the level of programming data structures.

In Figure 3 examples, concepts are anchored in an input/output, that is, stimulus/response. They form a sensorimotor feature space (colored regions) corresponding, for example, to different sensor modalities. Inherited features

(e.g., here, the penguin "is-a" bird and thus inherits the features of a bird) are shown with dotted lines, while red lines represent overwritten values (e.g., here, a penguin cannot fly). Green arrows point toward concepts that are themselves attributes of other concept features, accounting for inter-concept relationships. Values are derived from meta-information, not explicitly manipulated by the agent, but are used for process specification or interpretation (e.g., here, the weight unit and bounds).

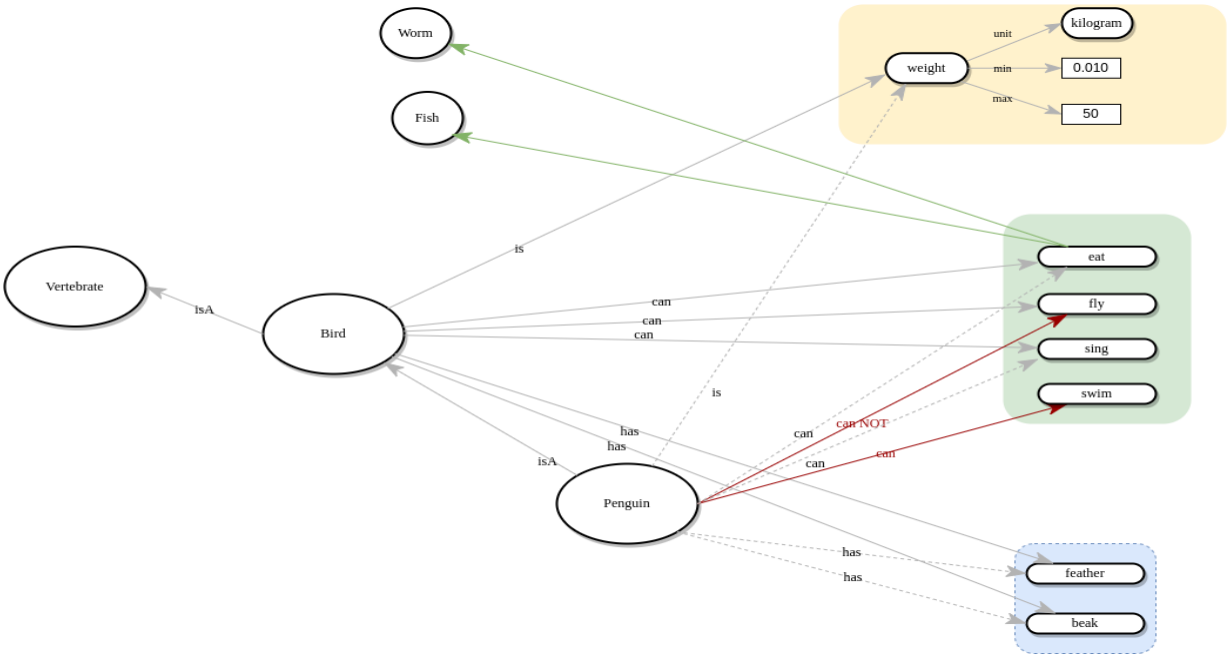


Fig. 3. Hierarchical data structure representing concepts. From [39].

This corresponds to the [32] approach, which also proposed the simple idea that an individual can be defined by "feature dimensions" (termed "quality" in the paper), corresponding to attributes with typed values. For instance, a bird's knowledge could correspond to what is represented in Fig 4. This general approach of semantic knowledge representation using a hierarchical taxonomy is instantiated considering (`is-a`) relations, with capability features (`can`), including those related to other resources, extrinsic features (`has`), and intrinsic features (`is`, as proposed by [38]). This is a typical framework, sufficiently general to encompass a broad range of domains in cognitive research. The illustrative example in Figure 4 is sufficient to allow us to detail the main characteristics of our representation. The textual representation here, and at the implementation level, is a weak form of the universal JSON syntax. Some features are properties, and others are relations. A property can be qualitative, e.g., the `is-covered-by` property takes a value in an enumeration (e.g., here, {`sing`, `fly`}), or quantitative (e.g., here, the `weight`). The features can be hierarchical, either because the value is an enumeration (e.g., here, `can`) or because the value has some features (e.g., here, `weight`).

Such a data structure defines a "concept" in the sense of [32] (e.g., here, "a bird") quoted before. It is a convex region of the state space (e.g., here, the area of all birds) in compliance with the formalism proposal [32]. This means that "between" to birds all intermediate element are also birds. This notion will be correctly formalized in the sequel. Each feature has also a default value, which defines the notion of "prototype" (e.g., here, a typical prototypical bird). Such hierarchical representation corresponds to the third cognitive memory architecture, as proposed by [24]. At the programming level, it will be implemented as a "type". At the geometric level, data values correspond to points, and concepts to "regions"; however, the data structure also encodes the prototype of a region.

When defining such a data structure, several design choices arise. On one hand, it is always better to decompose the information as much as possible into *atomic irreducible elements* (e.g., here, `family_name: Smith first_names: [John Adam]` instead of `name: 'Smith, John Adam'`) for algorithmic processing. On

```
bird: {
  is_a: vertebrate
  can: { sing fly eat: { worm fish } }
  has: { feather beak }
  is: { weight: { min: 0.010 max: 50 unit: kilogram } }
},
```

with some exceptions like penguins:

```
penguin: {
  is_a: bird
  can: { fly: false walk }
}.
```

Fig. 4. An example object defined via its features.

the other hand, it is preferable to organize features into substructures rather than present flattened information (e.g., here, create a substructure for the name, birth date, and related fields) to maximize modularity. We already mentioned the importance of providing as many default values as possible, and this is a design requirement at several levels; see, for instance, appendix A.2 for a discussion at the level of numeric data representation. When appropriate, defining bounds is also a precious meta-element. We also note, at the level of concrete implementation, that it is always preferable to use explicit, standard feature names that align with established terminology; thus, avoid acronyms or abbreviations and use the most common term for the feature.

3.2. Structuring the state space with data type

Given the previous computational objective, the first step is to generate a metrizable symbolic data-structure embedding (e.g., a "symboling"). In the resulting metric space, the lever is the notion of editing distance. As a result, a symbolic data value is step by step modified to equal another value, as detailed in the following subsection. Each elementary editing operation incurs an additive cost, yielding a well-defined distance and a minimal-distance path from the initial value to the target value. The key point is that such distance depends on the data type. For instance, given a numeric value, it will depend on the chosen bounds, scale, and precision, as developed in appendix A.2.

Moreover, given a data type, this specification requires defining a projection of data values in the neighborhood of the data type's region onto the data type. To precisely define these operations at the programming level, we base their implementation on the usual notion of *type*.

On one hand, atomic types correspond to string, numeric, or modal (a generalization of Boolean) values, as in any usual language, but here enriched meta-values, as shown in Table 1.

string	This data type specifies a subset of strings.
modal	This data type specifies a level of truth between -1 (false), 0 (unknown), and 1 (true), see appendix A.3.
numeric	This data type specifies a numeric value with its related metadata (bounds, precision, unit, ...), see appendix A.2.

Table 1

Basic data types.

On the other hand, compound types correspond to usual (i) enumeration, (ii) ordered lists, (iii) unordered sets, or (iv) records, as shown in Table 2. Records are sometimes called named tuples, or "object" in JSON syntax, "dictionary" in Python, "map" with a string key in other programming languages, and even "associative map" in different contexts, thus being a universal construct. Specification details and type parameterization are provided in Appendix A, along with a description of the related methods using standard algorithms.

enumeration	This data type specifies an enumeration of other values.
list-of-*	This data type specifies an ordered list of values.
set-of-*	This data type specifies an unordered set of values.
record	This data type specifies a record of values.
value	This data type is the root data type that corresponds to any value.

Table 2
Compound data types.

The present preliminary implementation only considers the basic atomic type, but it would be very easy to include all usual data types, used for instance in OWL. This includes structured data types such as (i) date, time and duration, (ii) IRI including URI, thus URL, (iii) geolocation, (iv) human language tags and locale, while (v) specific numeric data types with a given precision or sign, or (vi) numeric type extensions such as complex numbers or fixed dimensional vector or matrix. These types are structured data types. A string representation is also always defined with a given syntax, and we have observed that such syntax is always characterized by a regular expression, for the types enumerated in this paragraph. This means that there is a one-to-one mapping between such a string and the data record items. In our implementation:

- The RecordType implementation provides such a parsing mechanism to and from string representations.
- New types can be defined combining these ingredients, given specific parameters or deriving new types.

Based on this design choice, we consider a data structure to be the iterative combination of scalar and compound data types, and we will define editing distance and projection for such data structures.

3.3. Defining an editing distance

The editing distance is defined in terms of *editing operations*, each with a positive cost: insertion, deletion, or modification of an element. Each cost can be parameterized, allowing user-defined costs to model the data space and account for a priori application knowledge; we can weigh such costs to quantify the importance of a given feature. For instance, this means (l+) adding, (l-) deleting, or (l#) changing an element in a list, or (t+) introducing, (t-) deleting, or (t#) changing a value in a record, each of these operations having a user-defined positive cost. The distance is well-defined as the sequence of editing operations with a minimal cost, as shown in [1], for instance, where several alternatives are also proposed and compared:

- The distance of an element to itself is 0, because no editing operation is required.
- It is obviously symmetric, because each editing operation has an inverse: deletion versus insertion, or reverse change.
- The triangular inequality is also obvious because if you edit A to obtain B and then edit B to obtain C, you must not use fewer operations than to relate A directly to C.

This sequence of editing operations defined a (non-unique) editing path, with intermediate data structures at each step, making explicit which node has been added, deleted, or changed. Therefore, this mechanism not only defines a distance between two inputs but also provides a list of intermediate data structures between them. To the best of our knowledge, this is new with respect to editing distances.

Another critical point is that an editing sequence can be represented as a data structure, namely, an ordered list of editing actions.

At the data representation level, a complete data structure is a “forest”: A set of disjoint trees. Furthermore, the features of a given data structure are semi-ordered. This means that some are comparable, but not all. This is a significant property at the algorithmic complexity level, ensuring that the edit distance between two data structures has polynomial rather than exponential complexity, as developed and demonstrated in [49]. We also require that editing preserve the data structure type (e.g., a list cannot be converted to a set). This preserves the tree filiation, which makes it computable in polynomial time, as shown in [49], and we propose a variant of this approach here. If two data sets are of different, non-intersecting types, complete deletion followed by a complete insertion is the only admissible solution. Contrary to our design choices, in general, considering the editing distance in a tree as a

general graph, such as an ontology portion, is NP-hard, thus intractable [9], and sub-optimal techniques must always be considered [10].

Furthermore, the fact that we preserve data type makes the distance computation implementable using standard algorithms, as detailed in appendix A, for each data type, and listed now, considering Figure 5 as an illustration:

- For the record, each named value is linearly treated one by one, thus with a $O(N)$ complexity, since it corresponds to distinct type values.
  In the shown example, the name_1 is unchanged, thus at a zero distance; the name_2 value is to be deleted, which is equivalent to be changed to correspond to an empty value; the name_5 value is to inserted, changed from an empty value to the expected value; the name_3 and name_4 values are edited since the corresponding set and list are to be changed.
- For the ordered list, the well-known Levenshtein quadratic algorithm is used, thus with a $O(N^2)$ complexity, as described in [45].
  In the shown example, an insertion, the second item_2, and a modification of item_3 to item_5, allows to transform one list to another. Furthermore, two insertions of item_2 and item_5, and a deletion of item_3, constitute a second solution; the solution of lower cost is selected.
- For the unordered set, the well-known Hungarian cubic algorithm is used, described in [12], with a $O(N^3)$ cubic complexity.
  In the shown example, the deletion of element_1 makes the job.

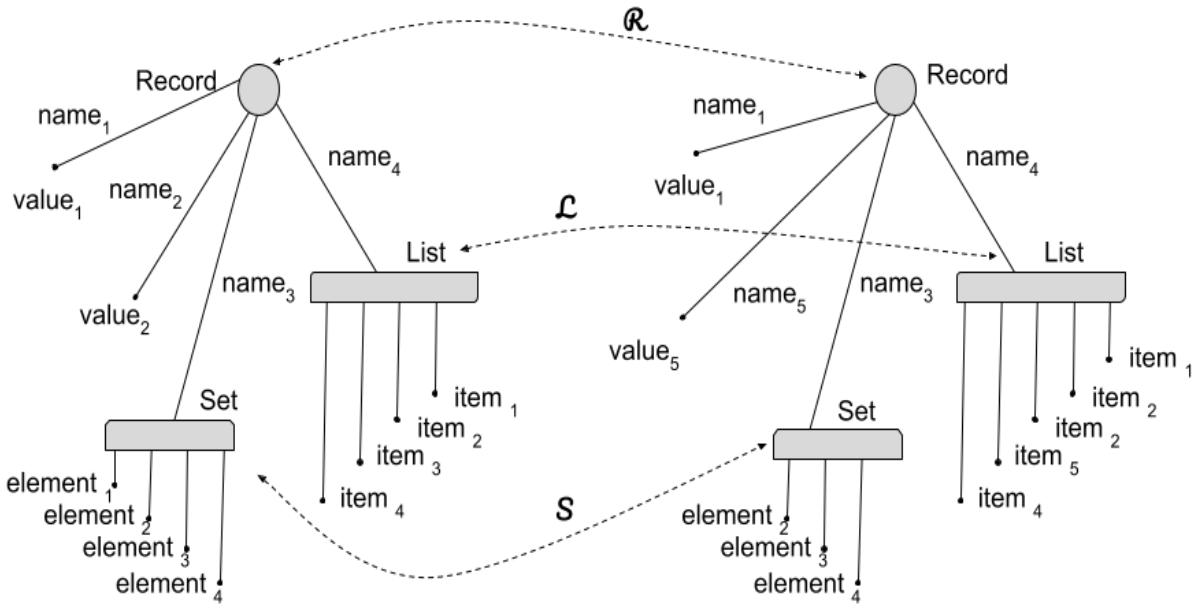


Fig. 5. The notion of edit distance, on an example including the main structured data types as illustration.

To illustrate such editing distance in a concrete, compelling case, let us consider in Figure 6 a creative open problem-solving experiment observation, using a pedagogic robotic activity, called "CreaCube" and presented in [53]. A temporal sequence of such states defines the activity representation as a trajectory from the subject's initial state to a final success or give-up state. Clearly, a suitable editing distance between states of resolution allows the representation of the subject's progression. More details are given in [39].

3.4. Local structuring of the metric space

We are in a metric space by virtue of the edit distance. This space is also equipped with a geodesic between two data structures, thanks to the nature of the editing distance. This path between two data structures $\mathbf{s}_1$ and $\mathbf{s}_2$

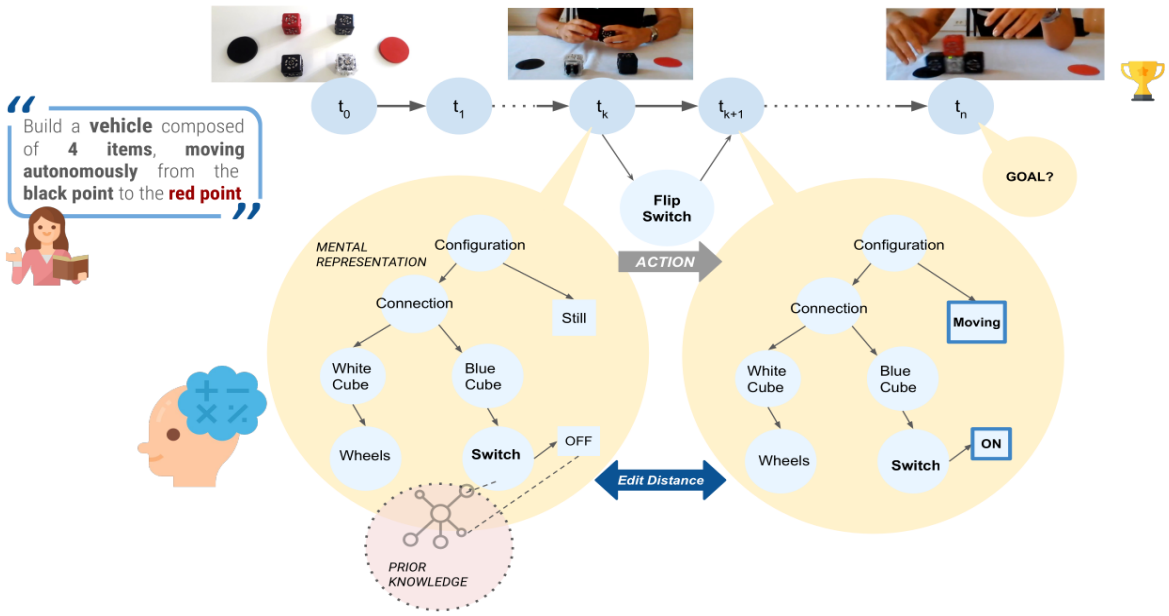


Fig. 6. An example of editing distance in a creative open problem solving experiment observation, from [39].

is a discrete sequence of intermediate data structures obtained by applying a minimal-cost sequence of editing operations to transform bfs_1 into bfs_2. The key point is that for any data structure $\mathbf{s}_k$ on this path, by construction of the editing distance[1]:

$$d(\mathbf{s}_1, \mathbf{s}_k) + d(\mathbf{s}_k, \mathbf{s}_2) = d(\mathbf{s}_1, \mathbf{s}_2),$$

which is the reason we call it a geodesic. A key point is that the path is not unique. To select one, in our case, at the implementation level, we have made the following default choices:

- For lists, including sequences, at equal cost we prefer edition to deletion, and deletion to insertion, so that we maintain or reduce the list length rather than increase it.
- For sets, corresponding to an unordered list, we first consider the association of maximal cost before the association of lower costs. Furthermore, we need to introduce a syntactic order among values with equal cost, as detailed below.
- For records, we consider the transformation of the item order, starting with the first item. This is consistent with the fact that, at the semantic level, item order is taken into account.

[1] **Intermediate editing configurations form a geodesic:** Let us consider between $\mathbf{s}_1$ and $\mathbf{s}_2$ an intermediate data structure $\mathbf{s}_k$ defined by a sub-sequence of editing operations that has been used to calculate $d(\mathbf{s}_1, \mathbf{s}_2)$.
-1- By construction, the cost $c(\mathbf{s}_1, \mathbf{s}_k)$ of the sub-sequence of operation from $\mathbf{s}_1$ to $\mathbf{s}_k$ plus the cost $c(\mathbf{s}_k, \mathbf{s}_2)$ of the sub-sequence of operation from $\mathbf{s}_k$ to $\mathbf{s}_2$ is equal to $d(\mathbf{s}_1, \mathbf{s}_2)$.
-2-The cost $c(\mathbf{s}_1, \mathbf{s}_k)$ can not be lower than the distance $d(\mathbf{s}_1, \mathbf{s}_k)$, because $d(\mathbf{s}_1, \mathbf{s}_k)$ corresponds to the cost of the lower sequence of operation from $\mathbf{s}_1$ to $\mathbf{s}_k$, by definition.
-3- This also the case from $\mathbf{s}_k$ to $\mathbf{s}_2$: we also have $c(\mathbf{s}_k, \mathbf{s}_2) \geqslant d(\mathbf{s}_k, \mathbf{s}_2)$.
Given -1-, -2-, and -3-, and we obtain:

$$\begin{cases} c(\mathbf{s}_1, \mathbf{s}_k) + c(\mathbf{s}_k, \mathbf{s}_2) = d(\mathbf{s}_1, \mathbf{s}_2) \\ c(\mathbf{s}_1, \mathbf{s}_k) \geqslant d(\mathbf{s}_1, \mathbf{s}_k) \\ c(\mathbf{s}_k, \mathbf{s}_2) \geqslant d(\mathbf{s}_k, \mathbf{s}_2) \end{cases} \Rightarrow d(\mathbf{s}_1, \mathbf{s}_2) \geqslant d(\mathbf{s}_1, \mathbf{s}_k) + d(\mathbf{s}_k, \mathbf{s}_2),$$

while from the triangular inequality we also have $d(\mathbf{s}_1, \mathbf{s}_2) \leqslant d(\mathbf{s}_1, \mathbf{s}_k) + d(\mathbf{s}_k, \mathbf{s}_2)$, so that we only must conclude about the equality.
Another argument is that this cost $c(\mathbf{s}_1, \mathbf{s}_k)$ can not be higher that the distance $d(\mathbf{s}_1, \mathbf{s}_k)$, otherwise it means that there is a "better" sub-sequence of operation from $\mathbf{s}_1$ to $\mathbf{s}_k$, which, without changing the sub-sequence of operation from $\mathbf{s}_k$ to $\mathbf{s}_2$, will lower $d(\mathbf{s}_1, \mathbf{s}_2)$, which is a contradiction. We thus must have $c(\mathbf{s}_1, \mathbf{s}_k) = d(\mathbf{s}_1, \mathbf{s}_k)$, with the same result for $c(\mathbf{s}_k, \mathbf{s}_2)$, so that we re-derive the equality again.

Such choices seem reasonable but are somehow arbitrary. At the semantic level, such decisions may have an impact, but they are easy to adapt to align with the intended meaning. A more efficient, but exponentially costly design choice, would have been to generate all possible paths of minimal distance between two data structures, implementing such a mechanism is easy but rapidly intractable when the data structure size increases.

A step ahead, it is easy to verify that for each data structure on such a geodesic, we have the minimal distance to both extremities, and the sub-geodesic segment corresponds to a geodesic of minimal length between the intermediate data structure and each extremity. We also readily verify that our design choice of a minimal geodesic is stable, in the sense that, in the three cases (list, set, and record), the geodesic segment corresponds to a geodesic that satisfies the corresponding uniqueness requirement.

A step further, this path is computed to obtain the minimum distance between two path elements. This is interesting because weighting the distance to satisfy application-dependent requirements allows us to select the geodesic among all possible choices. It is important to note that the type of a list or set must include the empty value when deleting or inserting, and this empty, undefined value is simply the default type value, its prototype.

With this notion of distance and geodesic, the symbolic data space is a kind of "non-differentiable manifold" (in an informal sense). Contrary to a usual abstract or embedded manifold, there is neither a local Euclidean tangent space nor any other kind of dense continuum defined in a neighborhood of a point, but only geodesic paths between points. As a consequence, a gradient of a scalar field is only defined in sparse directions, without any notion of opposite direction. This will have significant impacts in the sequel. Beyond its lack of utility for our developments, the theoretical study of such a metric structure would be of further interest.

3.5. Using data type to specify concepts

To structure the data, the basic construction is the notion of Type, as defined in computer science. The set of values of a given type defines a metric subspace, a region of the state space at the geometric level, equipped with

- (i) a distance between two values of the same type, as detailed before, and
- (ii) a projector from a neighborhood of this subspace onto it, as discussed in this subsection.

As a consequence, lists and sets must be lists or sets of the given type. This is not an obstacle: If, say, a list must contain elements of a finite set of types, then a super type including these types is to be defined. At the geometric level, this means a region including all kinds of areas. In this way, two points of different, disconnected types are always at a finite distance, by simply deleting the former and inserting the latter. The super region is thus connected, imposing that the empty value must belong to all data types, as mentioned above. It is easy to verify that, given this condition, such a union of regions is convex. This is an extreme case; if two types intersect, the geodesic between two points of different intersecting types need not include the empty value. A step ahead, all data values belong to a trivial super-type "value" with no further information, so that two values are at "zero" distance, yielding an empty geodesic, and they project to the "empty" value with no information. In practice, a derived data type is built from compound data types, with the specification of the type's parameters, for instance, numerical bounds and precision, or list minimal length, etc. It is itself a data structure, for which a vocabulary is defined, allowing to manipulate it a meta-level.

At the modeling level, this corresponds to a concept, as defined earlier and introduced by [32], and to the implementation design choices, detailed in appendix A. As expected by [32], a region is convex in the sense that along a geodesic between two values of a path, all values are of the same type. More precisely:

- By construction, scalar types, string, numeric value, or modal value form a convex region, because the intermediate values between two strings are obviously strings, and it is indeed also the case for numeric or modal values.
- Regarding compound type, since by construction of the editing distance, values are only related to values of the same type, convexity is also guaranteed.
- For user-defined types, thus defined by constraints on the previous types, the related distance must, by contract, only consider intermediate values of the same kind. For instance, intermediate values between two positive numbers are positive; intermediate values between two strings constrained by a regular expression must also be strings that verify this regular expression; and the same holds for derived compound types.

We thus see that type region convexity is a verified property for fundamental scalar and compound types, but a design requirement for derived types. It is also straightforward to verify that a region is a compact, complete, and path-connected metric space, owing to its convexity.

Furthermore, given two data structures, the editing distance is bounded, since transforming one into the other can always be achieved by deleting all features of the former and inserting all features of the latter. If the minimal editing distance equals this maximal bound, it means that both data structures have nothing in common. They are semantically disconnected, which also means that the empty data structure is on the shortest editing path. By introducing reasonable assumptions about the editing distance, this semantic connectedness defines a partition of the data structure space. Generally, we consider that modifying a value is less costly than deleting and inserting it, because deletion or insertion entails modifying the value to or from the empty value, which likely requires more operations than modifying a subset of the data elements.

The proposed representation can also borrow from the usual data-structure representation, namely, the notion of schema. A schema defines a set of data structures that verify constraints, for instance, whether a feature must be defined and, if so, which data type it has. This corresponds to well-established XML-Schema, or JSON-Schema. If a given data structure is compliant, its distance to the defined set is zero.

In our context, the notion is stronger. It not only defines a function whose value is actual if and only if it is compliant with the schema, but also a *projector*: a schema only defines a region of compliant data structures. If a given data structure is compliant, its distance to the defined region is zero. Otherwise, we propose defining a projector that maps a non-compliant data structure to the closest compliant data structure with respect to the editing distance, and also provides the corresponding editing sequence. More precisely, each required feature must have a default value, so that if missing, an insert operation can add it. Moreover, if a numerical value is out of bounds, projecting it onto the nearest bound provides a straightforward solution. Lists and sets can be empty by default or initialized to their default values.

Such a projection is a fundamental tool for manipulating these symbolic data structures at the geometric level. It is not apparent that such a projector can be implemented at the algorithmic level, and it must be specified for each data type. However, it appears that we can properly define it for all basic data types under consideration here, thanks to the notion of geodesic introduced previously and further detailed in appendix A.

3.6. Semantic and syntactic type regions

The projection of a value onto a given region is not necessarily defined everywhere in the data space; there are likely values that are unrelated to the data type. For such values, projecting is meaningless. By default, in such a case, projecting means deleting it and replacing it with the region *prototype*, which is defined solely by the default value. We thus consider:

- The *syntactic* super region of a type, where data values can be projected onto a valid value.
- The *semantic* region of a type, where data values are valid.

The key point is that, to compute a projection in a neighborhood of the semantic type region onto it, the value must satisfy certain syntactic constraints for the calculation to be well-defined. For instance, considering the type of positive integer, a value is syntactically valid if the string representation parses to a numeric value, and is semantically valid if this value is a positive integer. If the string parses to a numeric value, it is easy to define how to project such a real number onto a positive integer. Another simple example is that if a numerical value is out of bounds, projecting it onto the nearest bound provides a straightforward solution. As a counterexample, if a string can not be parsed as a number, then any numeric operation will be undefined. Moreover, if a feature is of the wrong type, the only solution is to delete the value and insert a default value instead.

A step further, the syntactic neighborhood may correspond to a more general super-type, where syntactically valid expressions are semantically valid concerning this super-type. As a result, the distance from a value to the type region can be computed using the super-type metric, provided that the projection corresponds to the shortest distance.

We also observe that, to obtain a unique path between two sets, all values of a given type must be ordered. Therefore, we also add this requirement, which may be merely syntactic or correspond to a semantic meaning. Strings are alphabetically ordered, numeric (including modal) values are obviously ordered, the element order orders

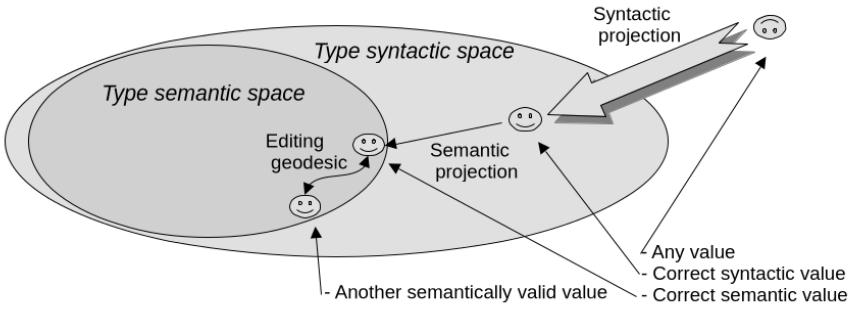


Fig. 7. Illustrating the notion of syntactic and semantic type.

lists, record name/value items are ordered, preserving by default the insertion order of the record keys, or sorted by any application-related importance of the record items. This allows to sort sets in their element order, and obtain an order for all basic and compound elements, while by derived type either use the default inherited order, or define a more suitable one.

3.7. Computing extrapolation away from a neighborhood value

Another issue is extrapolating the path from a value, away to another. Given a value $\mathbf{s}_0$, instead of finding a path towards a reference value $\mathbf{s}_\sharp$, as given by the geodesic path, we aim at creating a path "away" from this value. This is to be used, for instance, for exploration when learning a behavior, or to avoid an obstacle or a forbidden state when generating a plan or a trajectory. More generally, it is an interesting feature relative to finding "exploring" or "creative" solutions (in the broad sense).

The caveat is that we do not have a priori such a mechanism. Let us consider, as an example, the standard Levenshtein editing distance. It has no well-defined "inverse." If we consider the distance between `mama` and `mamie`, we easily obtain `mama` $\rightarrow$ `mame` $\rightarrow$ `mamie` in two steps. Defining a string edition `mama` that is "away from" `mamie` is ill-defined; we could insert, delete, or change any letter. Furthermore, considering the trivial example of words, there is a tiny chance that when inserting, deleting, or changing any letter, we draw a syntactically correct word, and even less chance to generate a semantically valuable word. More generally, for complex symbolic data structures, simply generating unconstrained random values around a current value via a simple random draw yields almost no chance of producing a useful or even meaningful value.

3.7.1. Using the data type bounds

We thus have to design this new functionality, given the existing ingredients. We explore the problem of choosing an interpolation from a data structure to a data-space bound, selecting the path that maximizes the distance to the corresponding data structure. Since, by construction, the data space is bounded as soon as it is sufficiently specified, as illustrated in Table 3, we can rely on these bounds.

Regarding regular expressions, of course, the accepted strings are not necessarily bounded. However, as soon as quantifiers are bounded (for instance $s*, max$, in words: s 0 to up to max times), the regular expression accepted strings are bounded. It is important to note that, given a hierarchical data structure, the number of bounds increases exponentially. A list or a set with *length* elements of a given type with *count* bounds, $count > 1$, will have $count^{length}$ possible bounds so that a list or set with $length \in \{minLength, maxLength\}$ elements will have

$$\sum_{length=minLength}^{maxLength} count^{length} = \frac{count^{Maxlength-1} - count^{Minlength}}{count-1}$$

bounds. A record with *length* items of type which bounds counts are $count_1 \cdot count_{length}$ will have

$$\Pi_{i=1}^{length} count_i$$

bounds.

string	Bounded if defined by either an enumeration or by a bounded regular expression or a *maxLenth* meta-value bound.
modal	Bounded in $[-1, 1]$.
numeric	Bounded as soon as the *min* and *max* meta-value are defined.
enumeration	Bounded by construction.
list-of-*	Bounded as soon as a *maxLength* meta-value is defined.
set-of-*	Bounded as soon as a *maxLength* meta-value is defined.
record	Bounded as soon as the record items to take into account are explicitly enumerated.
value	Bounded by the fact that it is only the undefined value as an explicit value.

Table 3
Basic and compound data bounds.

However, the goal is only to explore the space "away" for a given point. Because it is a search in an unknown part of the data space, it is not strictly defined. Therefore, to consider a manageable number of bounds, we may only draw a random subset of bounds.

3.7.2. Geodesic prolongation mechanism

It means that we have by construction bounds that can be used as escape values to be found on geodesics from the current value towards such bounds and away from the reference value, which is to be extrapolated, as illustrated in Figure 8. Choosing the best extrapolation path is an under-determined problem. Let us discuss how to specify it better.

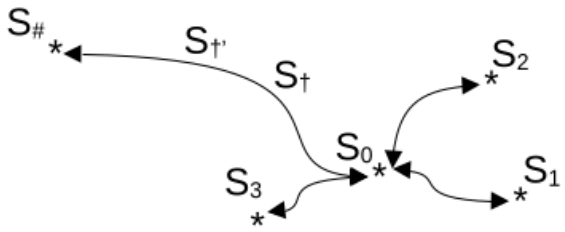


Fig. 8. Describing the extrapolation issue of a data $\mathbf{s}_i, i > 0$ through $\mathbf{s}_0$, given a bound $\mathtt{S}_\#$, suitable for $\mathtt{S}_1$ extrapolation, while less adapted to $\mathtt{S}_2$ and $\mathtt{S}_3$ extrapolation.

An interesting criterion, at the geometric level, is that geodesic from $\mathbf{s}_i$ to $\mathbf{s}_\sharp$ includes $\mathbf{s}_0$, so that $\mathbf{s}_0$ is an interpolation between $\mathbf{s}_\dagger$ and $\mathbf{s}_\sharp$, justifying the term of extrapolation, as a dual notion. This means, in terms of distances:

$$\nu_{i0}(\mathbf{s}_\dagger) \stackrel{\text{def}}{=} d(\mathbf{s}_i, \mathbf{s}_0) + d(\mathbf{s}_0, \mathbf{s}_\dagger) - d(\mathbf{s}_i, \mathbf{s}_\dagger) \simeq 0.$$

The quantity $\nu_{i0}(\mathbf{s}_\dagger)$ has the dimension of a distance, as a linear combination of distances. Obviously the problem of finding an extrapolation $\mathbf{s}_\dagger$ of $\mathbf{s}_i, i > 0$ through $\mathbf{s}_0$ in the direction of $\mathbf{s}_\sharp$ is meaningful only if distances between each of these two points are higher than 0.

In the general case, $\nu_{i0}(\mathbf{s}_\dagger) \geqslant 0$, due to the distance triangular inequality. We thus have some advantage to minimize $\nu_{i0}(\mathbf{s}_\dagger)$, to be as close as possible to the geodesic case. In fact [2] $\nu(\mathbf{s}_\dagger)$ increases with the distance $d(\mathbf{s}_0, \mathbf{s}_\dagger)$ to $\mathbf{s}_0$, while obviously $\lim_{\mathbf{s}_\dagger \to \mathbf{s}_0} \nu(\mathbf{s}_\dagger) = 0$.

[2] **Triangular inequality increases with extrapolation distance:** Let us consider two close $\mathbf{s}_\dagger$ and remote $\mathbf{s}_{\dagger'}$ extrapolation points on the $\mathbf{s}_0$ to $\mathbf{s}_\sharp$ geodesic, thus with:

$$d(\mathbf{s}_0, \mathbf{s}_\dagger) + d(\mathbf{s}_\dagger, \mathbf{s}_{\dagger'}) = d(\mathbf{s}_0, \mathbf{s}_{\dagger'}).$$

A few algebra yields:

But we also want the extrapolation to be as "away" as possible, which means that $d(\mathbf{s}_0, \mathbf{s}_\dagger)$ must be as high as possible; we thus have to balance both requirements and propose to maximize:

$$d(\mathbf{s}_0, \mathbf{s}_\dagger), \quad \nu_{i0}(\mathbf{s}_\dagger) \leqslant \epsilon_\mu$$

among all escape geodesic paths towards the chosen bounds $\mathbf{s}_\sharp$. We thus introduce a meta-parameter, ϵ_μ, that bounds the acceptable value of the geodesic triangle inequality.

3.7.3. Actual available implementation

The actual available implementation is based on bounds indexing:
- Only string explicit bounds enumeration is implemented at this stage, thus with obvious indexing, while numeric or modal have two bounds by construction.
- Given a hierarchical type (list, set, or record) of a given element type, an index is calculated modulo the number of element type bounds and indexing the position of the element in the hierarchical data structure.

We thus can directly return a bound by recursively decomposing this index for each element of the data structure. When randomly selecting a subset of bounds, thus a subset of indexes, we can
- either shuffle all indexes and select only the first ones,
- or draw indexes and check in a cache data structure that there is no repetition, the former method being more efficient if we select a large subset of indexes, the latter method being more efficient if we select a small subset.

3.7.4. Complementary specification of the extrapolation

A step further, to attempt to specify better what "away" could mean, and for a data structure $\mathbf{s}_\dagger$ on the geodesic between $\mathbf{s}_0$ and $\mathbf{s}_i$ we propose, as a perspective of this work, two other criteria.

On a first hand, on the distance, selecting only target $\mathbf{s}_\dagger$ with[3]:

$$d(\mathbf{s}_\dagger, \mathbf{s}_\sharp) > d(\mathbf{s}_\dagger, \mathbf{s}_0),$$

that is closer to the current data structure than the reference data structure.

On the other hand, on a generalization of the orientation, selecting only target $\mathbf{s}_\dagger$ with[4]:

$$d(\mathbf{s}_\dagger, \mathbf{s}_0)^2 + d(\mathbf{s}_0, \mathbf{s}_\sharp)^2 < d(\mathbf{s}_\dagger, \mathbf{s}_\sharp)^2,$$

e.g., $\mathbf{s}_1$ or $\mathbf{s}_2$ but not $\mathbf{s}_3$ in Figure 8, thus which a locally an obtuse angle and not an acute angle.

These distance- and orientation-complementary constraints help verify the coherence of the proposed solution or reduce the search along geodesics.

$$\begin{aligned}\nu_{i0}(\mathbf{s}_{\dagger'}) - \nu_{i0}(\mathbf{s}_\dagger) &= d(\mathbf{s}_i, \mathbf{s}_0) + \underbrace{d(\mathbf{s}_0, \mathbf{s}_\dagger) + d(\mathbf{s}_\dagger, \mathbf{s}_{\dagger'})}_{d(\mathbf{s}_0, \mathbf{s}'_\dagger)} - d(\mathbf{s}_i, \mathbf{s}_{\dagger'}) - d(\mathbf{s}_i, \mathbf{s}_0) - d(\mathbf{s}_0, \mathbf{s}_\dagger) + d(\mathbf{s}_i, \mathbf{s}_\dagger) \\ &= d(\mathbf{s}_i, \mathbf{s}_\dagger) + d(\mathbf{s}_\dagger, \mathbf{s}_{\dagger'}) - d(\mathbf{s}_i, \mathbf{s}_{\dagger'}) = \nu_{i\dagger}(\mathbf{s}_{\dagger'}) \geqslant 0\end{aligned}.$$

[3] **Extrapolation in the Euclidean case:** In the Euclidean case, for any point $\mathbf{v}_k \overset{\text{def}}{=} \alpha\,\mathbf{v}_0 + (1-\alpha)\,\mathbf{v}_i, \alpha \in [0,1]$ on the geodesic between $\mathbf{v}_0$ and $\mathbf{v}_\dagger$ which a rectilinear segment, if $d(\mathbf{v}_\dagger, \mathbf{v}_\sharp) < d(\mathbf{v}_0, \mathbf{v}_\sharp)$ we easily obtain:

$$d(\mathbf{v}_k, \mathbf{v}_\sharp) = \alpha\, d(\mathbf{v}_0, \mathbf{v}_\sharp) + (1-\alpha)\, d(\mathbf{v}_i, \mathbf{v}_\sharp) < \alpha\, d(\mathbf{v}_0, \mathbf{v}_\sharp) + (1-\alpha)\, d(\mathbf{v}_0, \mathbf{v}_\sharp) = d(\mathbf{v}_0, \mathbf{v}_\sharp),$$

thus we cannot obtain a extrapolation with $d(\mathbf{v}_k, \mathbf{v}_\sharp) > d(\mathbf{v}_0, \mathbf{v}_\sharp)$, but if $d(\mathbf{v}_\dagger, \mathbf{v}_\sharp) > d(\mathbf{v}_0, \mathbf{v}_\sharp)$ any $\alpha < 1$ allows to find an extrapolation. The assumption is that, in our case, for $\mathbf{v}_k$ close to $\mathbf{v}_0$, the data structure metric space is regular enough for this property to be approximately verified. However, the metric space does not enjoy an explicit property of "1st order Euclidean space", as for manifolds.

[4] **Orientation in the Euclidean case:** From the triangle cosine law:

$$d(\mathbf{v}_\dagger, \mathbf{v}_\sharp)^2 = d(\mathbf{v}_0, \mathbf{v}_\sharp)^2 + d(\mathbf{v}_\dagger, \mathbf{v}_0)^2 - 2\, d(\mathbf{v}_0, \mathbf{v}_\sharp)\, d(\mathbf{v}_\dagger, \mathbf{v}_0)\, \cos(\gamma), \gamma \overset{\text{def}}{=} \widehat{\overrightarrow{\mathbf{v}_0\mathbf{v}_\sharp}, \overrightarrow{\mathbf{v}_0\mathbf{v}_\dagger}},$$

and $d(\mathbf{v}_\dagger, \mathbf{v}_\sharp)$ is minimal, if $\gamma = \pi$, while obvious geometric properties of the triangle show that:

$$d(\mathbf{v}_\dagger, \mathbf{v}_\sharp) > d(\mathbf{v}_0, \mathbf{v}_\sharp) \Leftrightarrow \gamma > \tfrac{\pi}{2} \Leftrightarrow \cos(\gamma) < 0$$

which can be written only in terms of distance, in the case where $d(\mathbf{v}_0, \mathbf{v}_\sharp)$ and $d(\mathbf{v}_\dagger, \mathbf{v}_0)$ do not vanish:

$$d(\mathbf{v}_\dagger, \mathbf{v}_0)^2 + d(\mathbf{v}_0, \mathbf{v}_\sharp)^2 < d(\mathbf{v}_\dagger, \mathbf{v}_\sharp)^2,$$

while we still consider that this property could be locally reused in our case.

4. Defining Scalar fields of symbolic data structures

4.1. Position of the problem

When considering standard algorithms, such as variational approaches in supervised learning or clustering in unsupervised learning, we need to assign a numerical value to data structures, typically referred to as a cost or weight. Scalar fields allows to define several tools, such as:

- A cost function to optimize as used in many variational algorithms.
- A reward used in reinforcement algorithms.
- A trajectory potential used by some planning algorithms, as mentioned in the introduction.

Given a symbolic data structure $\mathbf{s} \in \mathcal{S}$, where $\mathcal{S}$ stands for a set of data structures, finite but huge and thus intractable to enumerate, a scalar field f is a real-valued function:

$$\begin{aligned} f : \mathcal{S} &\hookrightarrow \mathcal{R} \\ \mathbf{s} &\to r \end{aligned}$$

while, in our context, we define the function by setting some values:

$$f(\mathbf{s}_1) \leftarrow r_1, f(\mathbf{s}_2) \leftarrow r_2, \cdots$$

and would like to infer values for data structures in a neighborhood of these predefined values.

4.2. Neighborhood interpolation

Given a data structure $\mathbf{s}_0$ for which the scalar value has not been defined, we need to interpolate this value given known values in a neighborhood. Given the fact that we only have a metric space equipped with a distance, we define such a neighborhood with two parameters:

- Its cardinality K: we consider at most the K closets data structures.
- A maximal distance $d_{\max}$: we consider data structures whose distance is lower than $d_{\max}$.

This is a very standard way of defining neighborhoods, for instance, when considering operations on manifolds, as developed in [13], or in the context of dimensional reduction or graph manipulation, as used by [50] in the context of graph clustering, quoting here two very different examples to illustrate the use of such a neighborhood definition.

Given such a neighborhood, three cases are considered:

- If $K = 0$ the value is undefined.
- If $K = 1$, we can only approximate the data structure scalar value by considering the closest known predefined value, which is the unique data structure in this neighborhood singleton.
- If $K > 1$, we propose to iteratively reduce the neighborhood to a singleton, with $K = 1$. We consider, in the initial neighborhood, the two data structures whose relative distance is minimal and replace the two data structures with a data structure on their geodesic which is as close as possible to $\mathbf{s}_0$, thus reducing the neighborhood size by 1, as explained in Figure 9.

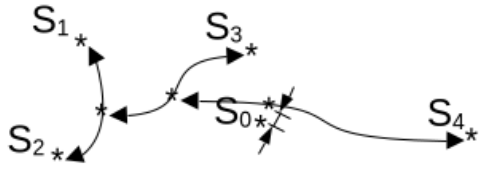


Fig. 9. As graphically represented here, $\mathbf{s}_1$ and $\mathbf{s}_2$ are replaced by the value on their geodesic as closed as possible to $\mathbf{s}_0$, which is then related to $\mathbf{s}_3$ by a geodesic, making the same choice again, which finally is related to $\mathbf{s}_4$, yielding to a close approximation to the unknown $\mathbf{s}_0$ value.

Given this mechanism for two data structures $\mathbf{s}_i$ and $\mathbf{s}_j$ and choosing as best location for the approximation, the data structure $\mathbf{s}_\bullet$ on the geodesic which distance to $\mathbf{s}_0$ is minimal, we can interpolate the corresponding scalar value

$r_\bullet$ by a linear interpolation:

$$r_\bullet \leftarrow \frac{d(\mathbf{s}_i,\mathbf{s}_\bullet)\, r_j + d(\mathbf{s}_j,\mathbf{s}_\bullet)\, r_i}{d(\mathbf{s}_i,\mathbf{s}_\bullet)+d(\mathbf{s}_j,\mathbf{s}_\bullet)}.$$

This value is well defined as soon as the three points $(\mathbf{s}_i, \mathbf{s}_j, \mathbf{s}_\bullet)$ are not equal, while:

$$\mathbf{s}_i \neq \mathbf{s}_j \Rightarrow 0 < d(\mathbf{s}_i, \mathbf{s}_j) < d(\mathbf{s}_i, \mathbf{s}_\bullet) + d(\mathbf{s}_j, \mathbf{s}_\bullet).$$

Choosing to merge the closest data structures and replace them with an intermediate structure that approximates their values is a reasonable approach in this context and yields a linear-time algorithm. At this stage, our method is suboptimal because it considers only one geodesic rather than all possible geodesics. At the cost of a combinatory explosion, we could also have considered all possible geodesics between all pairs of data structures and all possible sub-geodesics between the data structures on each geodesic, to attempt to obtain a data structure as close as possible to $\mathbf{s}_0$, but this would not have been a tractable design choice. Of course, these remarks also apply to extrapolation, since it is nothing more than interpolation with respect to bounds.

We cannot explicitly assume that the data structures live on an abstract manifold, a space that is locally Euclidean up to first order. However, let us propose a method inspired by such formalism. This method may be compared with a straightforward linear approximation:

$$r_0 \leftarrow \frac{\sum_{i=1}^{K} \nu(d(\mathbf{s}_0,\mathbf{s}_i))\, r_i}{\sum_{i=1}^{K} \nu(d(\mathbf{s}_0,\mathbf{s}_i))}$$

where $\nu(d)$ is some decreasing function of the distance, for instance: $\nu(d) \stackrel{\text{def}}{=} e^{-\frac{d}{d_{\max}}} \simeq 1 - \frac{d}{d_{\max}}$. The advantage of the former method is that it is more local: we perform linear approximation only between pairs of data structures that are as close as possible, and we consider only geodesics entirely contained within the region under consideration. The latter method relies on an arbitrary kernel $\nu(d)$, considers points that may be farther from the point to be interpolated, thereby introducing greater bias, and does not guarantee that the point is included in the region under consideration.

Another interesting point is that the same iterative algorithm can be adapted to approximate a barycenter between data structures.

4.3. Computing the barycenter of values

Given a set or subset of weighted data structures $\{(\mathbf{s}_1, w_1), (\mathbf{s}_2, w_2), \cdots\}$ we would like to approximate a data structure $\mathbf{s}_0$, such that[5]:

$$\mathbf{s}_0 \simeq \arg\min l, l \stackrel{\text{def}}{=} \sum_{i=1}^{K} w_i\, d(\mathbf{s}_0, \mathbf{s}_j)^e, e > 0, \forall i, w_i \geqslant 0$$

which corresponds to a generalization of a barycenter, as it is required for instance in clustering algorithms, such as K-means clustering.

Let us again consider three cases:

- If $K = 1$, obviously, $\mathbf{s}_0 = \mathbf{s}_1$ is the solution since we have a minimum when $d(\mathbf{s}_0, \mathbf{s}_1) = 0$.
- If $K = 2$, it is a reasonable choice to minimize, for a given exponent e, on the geodesic between $\mathbf{s}_1$ and $\mathbf{s}_2$:

$$e > 0, l_e = w_1\, d(\mathbf{s}_0, \mathbf{s}_1)^e + w_2\, d(\mathbf{s}_0, \mathbf{s}_2)^e.$$

[5] **Barycenter in the Euclidean case:** In the Euclidean case, $d(\mathbf{v}_0, \mathbf{v}_j) \stackrel{\text{def}}{=} \|\mathbf{v}_0 - \mathbf{v}_j\|$ and with $e = 2$, we easily obtain for such a convex quadratic criterion by derivation of the normal equations:

$$\mathbf{v}_0 \stackrel{\text{def}}{=} \frac{\sum_{i=1}^{K} w_i\, \mathbf{v}_i}{\sum_{i=1}^{K} w_i},$$

obtaining the usual formula of a barycenter, or weighted centroids of the case with data structures, where we can only consider distances.

On one hand [6] there is a unique minimum $\bar{l}_e$, found scanning from $\mathbf{s}_{\bar{i}}, \bar{i} = \arg\min_i w_i$, to the other geodesic end, to find the 1st local minimum of l_e, while if $e \leqslant 1$, the minimum is at one extremity of the geodesic, which is not an interesting balanced estimation.
On the other hand [7], the global minimum is on the geodesic.

– For $K > 2$, as before we can only work on geodesics and we propose to consider the two data structures $\mathbf{s}_i$ and $\mathbf{s}_j$ with the smallest distance, calculate on their geodesic the barycenter $\mathbf{s}_\bullet$ with weight $w_\bullet = w_i + w_j$, replacing the two former data structures by the latter, thus decreasing the set of data structure cardinal by 1, this being iterated until $K = 2$.

This is again only a suboptimal strategy, since we consider only a linear number of geodesics rather than all possible ones. Still, it provides an approximate estimate using only local operations.

5. Illustrative examples

At this preliminary stage of development, we are not yet able to report large-scale experimentation. Still, we would like to share several illustrative toy experiments that already allow us to evaluate and illustrate the proposed mechanisms.

---

[6] **Minimum as a function of the exponent:** Let us consider:

$$\begin{aligned}
\bar{\mathbf{s}}_0 &\stackrel{\text{def}}{=} \arg\min_{\mathbf{s}_0} \; w_1\, d(\mathbf{s}_0,\mathbf{s}_1)^e + w_2\, d(\mathbf{s}_0,\mathbf{s}_2)^e \\
&= \arg\min_{\mathbf{s}_0} \; w_1\, d(\mathbf{s}_0,\mathbf{s}_1)^e + w_2\, \left(d(\mathbf{s}_1,\mathbf{s}_2) - d(\mathbf{s}_0,\mathbf{s}_1)\right)^e && \text{because on a geodesic} \\
&= \arg\min_{d_{01}} \underbrace{w_1\, d_{01}^e + w_2\, (d_{12} - d_{01})^e}_{\stackrel{\text{def}}{=} l_e(d_{01})} && \text{writing } d_{ij} \stackrel{\text{def}}{=} d(\mathbf{s}_0,\mathbf{s}_1) \text{ for convenience.}
\end{aligned}$$

Since $\mathbf{s}_0$ is on the geodesic from $\mathbf{s}_1$ to $\mathbf{s}_2$, we have $d_{01} \in [0, d_{12}]$. In fact, d_{01} takes discrete values along the geodesic, but we begin by studying $l_e(d_{01})$ as a real-valued function. We also assume, without loss of generality, $w_1 \leqslant w_2$.
- If $e \leqslant 1$: the minimum is on one geodesic extremity, because:

$$\begin{aligned}
l_e(d_{01})|_{\mathbf{s}_0=\mathbf{s}_2} &= w_1\, d(\mathbf{s}_1,\mathbf{s}_2)^e \\
&= w_1\, [d(\mathbf{s}_1,\mathbf{s}_0) + d(\mathbf{s}_0,\mathbf{s}_2)]^e && \text{because on a geodesic} \\
&\leqslant w_1\, d(\mathbf{s}_1,\mathbf{s}_0)^e + w_1\, d(\mathbf{s}_0,\mathbf{s}_2)^e && \text{because } d \to d^e, e \leqslant 1 \text{ is convex} \\
&\leqslant w_1\, d(\mathbf{s}_1,\mathbf{s}_0)^e + w_2\, d(\mathbf{s}_0,\mathbf{s}_2)^e && \text{because } w_1 \leqslant w_2 \\
&= l_e(d_{01})
\end{aligned}$$

showing that $\bar{\mathbf{s}}_0 = \mathbf{s}_2$ in this case.
- If $e > 1$, since $l_e(\cdot)$ is derivable with $l'_e(d_{01}) = e\, \left(w_1\, (d_{01})^{e-1} - w_2\, (d_{12} - d_{01})^{e-1}\right)$ we must have at an extremum:

$$\begin{aligned}
& l'_e(\bar{d}_{01}) = 0 \\
\Leftrightarrow \quad & w_1\, (\bar{d}_{01})^{e-1} = w_2\, (d_{12} - \bar{d}_{01})^{e-1} \\
\Leftrightarrow \quad & w_1^{\frac{1}{e-1}}\, \bar{d}_{01} = w_2^{\frac{1}{e-1}}\, (d_{12} - \bar{d}_{01}) \\
\Leftrightarrow \quad & \rho\, \bar{d}_{01} = d_{12} - \bar{d}_{01} && \text{writing } \rho \stackrel{\text{def}}{=} \left(\tfrac{w_1}{w_2}\right)^{\frac{1}{e-1}} \\
\Leftrightarrow \quad & \bar{d}_{01} = \tfrac{d_{12}}{1+\rho}
\end{aligned}$$

This means that there is only one extremum. It is easy to verify that it is a minimum on $l'_e(\cdot)$, since $l'_e(0) = -w_2 (d_{12})^{e-1} < 0$ and $l'_e(d_{12}) = w_1 (d_{12})^{e-1} > 0$ with one change of sign at $\bar{d}_{01}$, only. It thus decreases and then increases. Furthermore, we easily verify, for a given $e > 1$, that $\rho \in]0, 1]$, the left bound corresponding to $w_1 \to 0$ and the right bound to $w_1 = w_2$ so that we obtain:

$$\tfrac{d_{12}}{2} \leqslant \bar{d}_{01} < d_{12}.$$

This means that the extremum can not be at $\bar{\mathbf{s}}_0 = \mathbf{s}_2$ (otherwise we would have $\bar{d}_{01} = d_{12}$). We also verify that

$$\lim_{e\to 1} \rho = 0 \Rightarrow \lim_{e\to 1} \bar{d}_{01} = d_{12} \Rightarrow \lim_{e\to 1} \bar{\mathbf{s}}_0 = \mathbf{s}_2,$$

in words: the minimum gets closer and closer to the geodesic bound, when $e \to 1$.

[7] **Minimizing on the geodesic is optimal:** Let us consider a general data structure $\mathbf{s}$, with a fixed distance $d_1 = d(\mathbf{s},\mathbf{s}_1)$ to $\mathbf{s}_1$, and $e > 0$. Then:

$$l = w_1\, d_1^e + w_2\, \left(d_1 + d(\mathbf{s}_1,\mathbf{s}_2) - \nu_{12}(\mathbf{s})\right)^e$$

where $\nu_{12}(\mathbf{s}) \stackrel{\text{def}}{=} d(\mathbf{s},\mathbf{s}_1) + d(\mathbf{s},\mathbf{s}_2) - d(\mathbf{s}_1,\mathbf{s}_2)$ as already used previously. Obviously l is minimal when $\nu_{12}(\mathbf{s}) = 0$, that is when $\mathbf{s}$ is on the geodesic. In other words, for any data structure, there is a better solution on the geodesic, as expected.

5.1. Morphing experiments

Given two symbolic data structures, computing the editing distance between them and the related geodesic path, very naturally generates what is called a morphing in image processing. This is particularly interesting, as it allows us to visualize geodesic paths in some examples, as reported in [7].

5.1.1. Visual morphing demo

A image morphing demo, using the SVG symbolic description of a drawing, allows one to reproduce pixelic image morphing, for some vectorial drawing, as shown in Figure 10. At the validation level, the present formalism can also be applied to realistic open standards, such as SVG. What the geodesic path explicitly represents is one (or several) optimal sequence of drawing-editing operations that transforms the former drawing into the latter. The subset of the SVG standard in use in this demo is defined as a symboling type. Given this, vectorial drawings are manipulated in the same way as other data structures. Please refer to the demo link for a detailed observation of the produced image sequence. This preliminary result shows that the symbolic data structure defining the drawing elements changes primarily from top to bottom, reflecting the record-item ordering, which is an arbitrary choice induced by the geodesic selection among all possible geodesics. As mentioned previously, the fact that user-defined parameters weight the editing distance, and that the geodesic path is defined by a local change of minimal editing cost, will readily allow a user to tune the geodesic choice. The intermediate color shows that the color space is also related to a distance in the SVG specification. The editing distance cost is 1965, with each elementary operation counted as 1, which appears high relative to the editing operations, such as changing or moving a drawing element. This is however reasonable because each editing gesture at "human" level corresponds in fact to a rather large set of change in the data structure elements. It also shows that, for realistic data sets, the number of elementary operations may become very large.

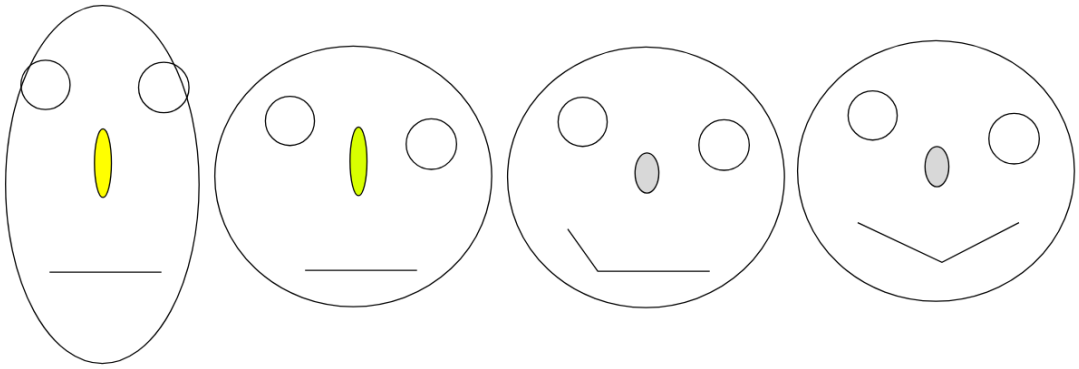


Fig. 10. A subset of the morphing between the "titi" drawing on the left and the "toto" drawing on the right: two intermediate images on the geodesic path defined by the SVG drawing representation are shown.

5.1.2. Audio morphing demo

A step further, the same collaborators have used our software to explore what could be a "musical morphing" at the symbolic level. Considering the first bars of two very popular kid songs (at least in France :) ...), and based on the MIDI standard, also translated as a symboling type for the standard subset used here, they have reproduced the same calculation, as shown in Figure 11, while a complete demo result is available on the demo link. Since the former song uses tied notes, while the latter uses dotted notes, it easy to see which part of the scores turns from the latter to the former, along the chosen path: it definitely shows that the default choice is reasonable but arbitrary (here, from the beginning to the end) while it could have been other way round, or something else. The editing distance cost is 37220, with each elementary operation counted as 1, yielding an average of about 450 operations between two consecutive elements on the path. Again, this may seem dizzying, but this is at the scale of this not-so-trivial data set.

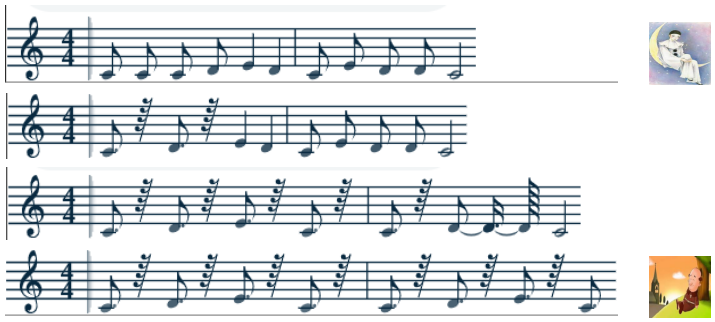


Fig. 11. A subset of the morphing between the “au clair de la lune” and the “frère jacques” early parts of the songs: two intermediate scores on the geodesic path defined by the MIDI music representation are shown.

5.2. *A symbolic problem solving setup*

Beyond simply observing the path between two data structures, we have also considered stating a problem-solving setup. At the demonstration level, the idea was to model a “climbing problem solving”: how to choose the next hold when climbing a wall? At our validation level, the goal is to assess the feasibility of modeling both the climbing wall and the climber, subject to constraints on limb length and posture. The result is shown in Figure 12. The complete experimentation available online allows the toy setup to be connected to an external trajectory-resolution algorithm, here using a standard `Python` genetic algorithm. This, in turn, shows that the middleware developed in `C/C++` is readily usable from `JavaScript` at the web interface level and from `Python` at the algorithmic and data analysis levels. Although still preliminary, this illustrates how we can address problem-solving issues at the symbolic level.

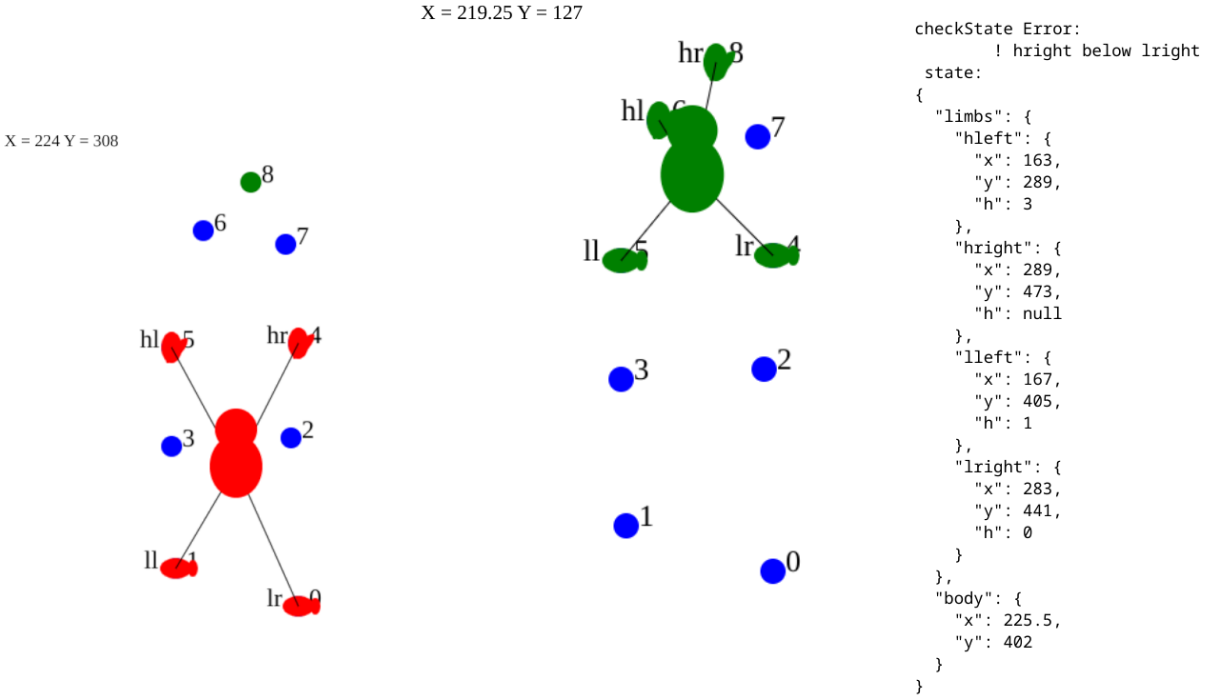


Fig. 12. The climbing experiment on a wall. Left: the avatar in some intermediate state. Middle: a possible final position. Right: The climber state in JSON syntax, with a detection of a rejected previous tentative of posture error (the right hand can not be below the right foot in this setup).

5.3. Reinforcement learning in a survival experiment

Here, we consider the application of standard machine learning algorithms, such as reinforcement symbolic learning mechanisms, as introduced in [41]. Reinforcement learning is a subfield of machine learning concerned with how agents learn to make decisions in an environment to maximize a notion of reward. It has shown great promise in a variety of domains, but it often struggles with generalization and interpretability due to its reliance on black-box models. One approach to address this issue is to incorporate knowledge representation into reinforcement learning, which is a key advantage of such a symbolic approach.

In [52], reviewed here, the authors explore the benefits of using symbolic representation in reinforcement learning and present preliminary results on its performance compared to standard reinforcement learning techniques. The agent chooses an action among a discrete set (e.g., eat, attack), and the agent's parameters are updated accordingly. The overall goal of the agent in this tiny reinforcement learning example is to survive. The add-on of this work is to consider a simple but formal ontology, as represented in Figure 13, and, as far as the symbolic description of food is concerned, in Figure 14. The agent feels internally its mood, health level, and hunger, each with a value ranging from negative (e.g., sadness) to positive (e.g., joy). It involves the senses of sight, taste, touch, and smell, and is defined by qualitative values. The external environment influences internal states as a function of the encountered entities. The key point is that, even in such a simple setup, each sensory cue is not a simple enumeration of values; rather, it is a relation among them and with external objects, necessitating the model's implementation as a hierarchically structured information. The ontology structures all entities, states, and actions within a class taxonomy, enabling the reuse of learned knowledge for related elements within a given neighborhood. There are 256×25×9 = 57600 possible values (number of internal states × number of entities × number of actions). This does not correspond to a standard benchmark in the literature, but it would be an interesting perspective on this work to consider such a case and compare the results with state-of-the-art techniques. The implementation is available as documented open-source software, and the software interface is textual, using prompts.

An a priory analysis yields that:

- It should take more than 10^{8-9} steps for a classical standard reinforcement learning algorithm to visit thus evaluate at least once each possible values to learn the expected reward and thus generate a correct strategy, whereas
- The symbolic approach should have better generalization capabilities and need far fewer steps in the environment to obtain a good reward, because the information can be shared between related elements.

In this work, the authors consider the widely used Q-learning algorithm at the level of numerical implementation using the well-established AI Gym, now Gymnasium middleware. They have implemented the [41] symbolic Q-Learning algorithm, which computes the editing distance, as developed here, to evaluate the reward at a point as an exponentially weighted sum of known rewards in the neighborhood, thereby leveraging the state-space metric. The key meta-parameter for symbolic learning is the neighborhood radius ρ, which controls the exponential decay.

The results shown in Figure 15 show that the use of symbolic representation can significantly improve the generalization capabilities of reinforcement learning agents, provided that the neighborhood size is well adjusted:

- If too small (e.g., 0.01), the related information is not correctly taken into account.
- If too large (e.g., 0.06), too general information is taken into account, and the estimation is biased.
- An appropriate radius (e.g., 0.04) yields a significantly better performance, this quickly (after about 2000 steps, in comparison to the standard approach that has not started to converge after 10000 steps).

However, as the authors conclude, the ρ hyperparameter, which scales the editing distance, is highly sensitive, leading to substantial changes in performance. Therefore, exploring ways to dynamically adjust the radius parameter over time to learn the optimal radius value is warranted. In fact, the issue is more general: algorithms using the current approach are highly dependent on the design of the edit distance metric.

6. Discussion

6.1. On the computational efficiency of the method

The data structure is implemented in `C/C++` using a data object model based on the `std::unordered_map` associative container, with a `std::vector` for record keys. We obtain performance comparable to that of N.

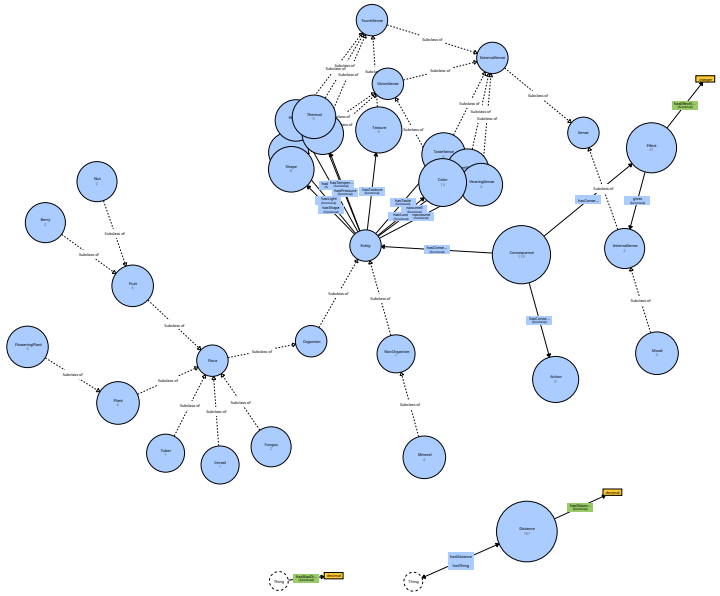


Fig. 13. The ontology used to describe a simplified external and internal environment in which an agent survives, from [52].

Lohmann C++ native library written in `modern C++`, as shown in Table 4. The test has been conducted on three Milo Yip benchmark sets: `twitter` a 13913 attributes structured JSON structure, `canada` a 167178 attributes multi-dimensional numeric array, and `cimp` a 37777 attributes database JSON structure. The present implementation is for some operations a bit slower because the record item insertion order is maintained, for semantic reasons detailed in this paper. However, this overload is negligible relative to the editing distance computation presented next. Both middlewares are less performant than speed-optimized libraries such as jsoncpp, which provide minimal functionality. The CPU computation times are evaluated on a standard mid-range laptop with `Intel©Core`^{TM} `i5-8265U CPU @ 1.6GHz x 8` with `16 GBiB memory` and no GPU usage.

	twitter			canada			citm		
	read	write	copy	read	write	copy	read	write	copy
lohmann	78	25	3	361	60	15	144	24	7
wjson	44	23	9	357	126	124	75	39	24

Table 4

Comparing CPU computation time, in milliseconds, between the Lohmann and our data structure operations, for three representative benchmarks.

Beyond data read and write access, geodesic path computation after distance estimation accounts for the majority of computational time. For the morphing demos, we obtain the results shown in Table 5.

Edition distance computation scales superlinearly with the involved data structures; linearly with respect to records; and with quadratic and cubic increases for lists and sets, respectively, whereas path computation scales

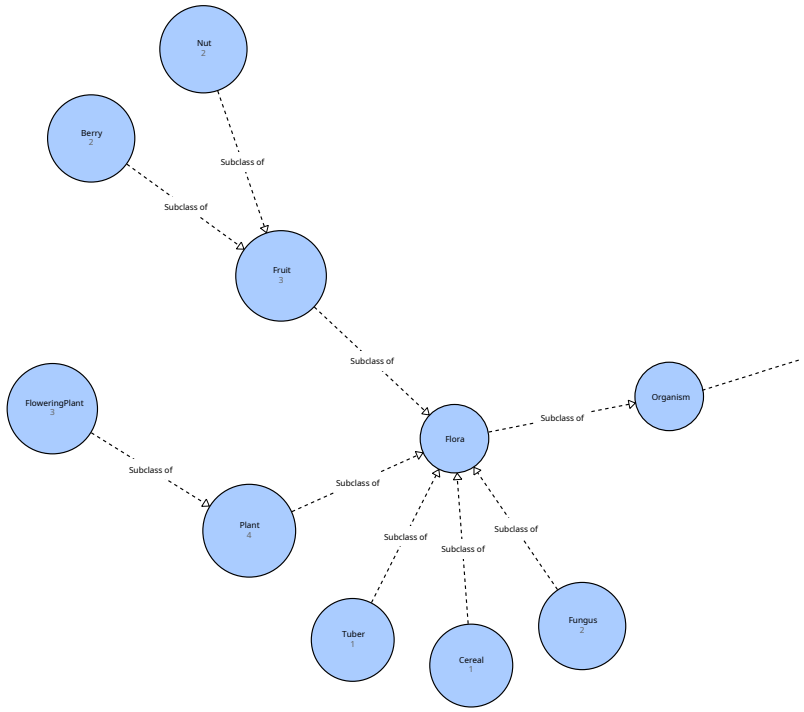


Fig. 14. A detail of the ontology regarding the symbolic description of food, from [52].

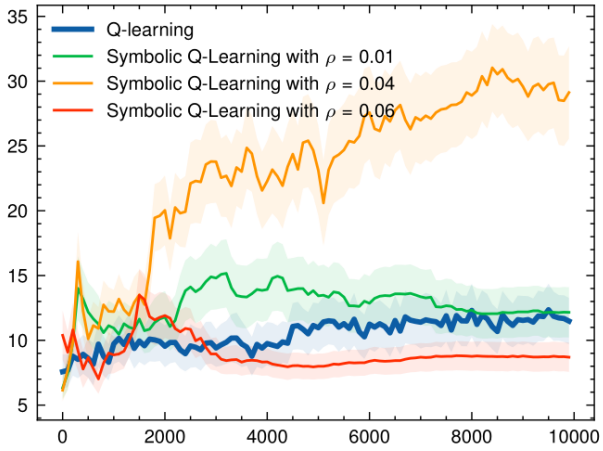


Fig. 15. Comparison between a Q-Learning and a symbolic Q-Learning cumulative rewards, considering several neighborhood sizes for the latter, from [52].

	path length	element attribute count	path attribute count	distance msec computation time	ratio $\frac{msec}{attribute}$	path msec computation time	ratio $\frac{\mu sec}{attribute}$
visual morphing demo	28	97	2724	330	3	473	173
musical morphing demo	80	170	14176	423	2	3293	232

Table 5
Path and distance CPU computation times, in two cases.

linearly with its length and the data structure attribute count. The small experimental observation confirms that adding the computation of the geodesic path to the standard edit distance computation yields only a 5% to 10% overhead.

Regarding memory consumption, using an associative map and an index vector scales linearly with the number of attributes. The element overhead of such an associative map, coupled with an index vector, is two memory slots per attribute and a few memory slots for the related record overhead. In the current implementation, the caveat is that strings are managed by copying rather than by reference, using a string dictionary. This would be a noticeable improvement, and it is a close perspective of this work.

These numbers have no "absolute" meaning. They show that the proposed computations have an order of magnitude comparable to what can be performed on a modern laptop, without requiring large computer clusters or dedicated hardware. This also provides a rule of thumb for estimating the computational cost of problems at higher scales.

6.2. Advantages and disadvantages of such metric space embedding

Some comparisons between the advantages and disadvantages of the proposed metric space and other grounding approaches have already been sketched throughout the paper. Let us synthesize the main elements.

Currently, symbolic data, such as text, are embedded in parametric vector spaces using statistical measures, and the resulting data are processed by black-box artificial neural networks with architectures such as generative deep neural networks, including variational transformers. This enables the attainment of unprecedented information-processing capabilities, but at substantial financial and environmental costs, raising several concerns (tech giant domination, intellectual property, ...).

The proposed approach avoids the initial embedding of symbolic data in a large-dimensional vectorial space. Conversely, it directly equips the symbolic set with the required numerical tools (metric, path, projector, ...) to apply standard machine learning algorithms, such as clustering or trajectory generation, as illustrated previously. We thus expect such an approach to be more parsimonious, yielding results that are easily interpretable and explainable, in the sense of [16]. We also noted that it would be more explanatory when applied to brain modeling, as discussed by [57].

Another practical aspect is ergonomic. Our experience in multidisciplinary research (here, learning science, neuroscience, and computer science), such as during the AIDE exploratory action, allowed us to verify that:

- The notions of state data point, distance between two points, step by step minimal distance path (the geodesic), data space regions organized in a hierarchy (the types), attaching a cost to each point (the scalar field), are easy to share when technical aspects are cleaned up.
- The extensive use of JSON syntax, especially if made more flexible with the wJSON dialect, makes specification easily readable and editable, without any need for deep practice.

In addition, the present effort to develop open, multi-platform, multi-language middleware should also help.

However, in practical applications, the proposed distance depends on numerous somewhat arbitrary parameters that are likely to influence the algorithm's performance. This has been experimented with in subsection 5.3 for the global distance scale ρ. More generally, the modeling choices and the number and quality of properties associated with each data type are crucial for deriving a pertinent specification. This questions the applicability of the proposed metric space.

What has been done in subsection 5.3 may help solve this caveat, and leads to a kind of recipes:

- Following, for instance, the tutorial presentation of [47], one may start modeling and designing the problem description, enjoying the quality and richness of the ontology approach, restricting as much as possible to the RDFS level of description of class and property hierarchies, for both simplicity and efficiency.
- From such a distributed specification, the next step is to study to what extent this could be structured in a hierarchical way with record, list, and set, noticing that such structuring is fully available at the ontology level.
- For each quantitative parameter, one should account for all relevant metadata, such as bounds and precision, as developed in appendix A.2; this will normalize all numerical elements when computing the edit distance.
- Ideally, by default, each data type assigns equal weights to its elements, and a default editing distance can be computed without having to preset each parameter. Then, if some a priori knowledge allows one to modify this default choice, that could be easily done, preferably with parsimony.

Following such a track may help in designing and parameterizing the data structure. As an alternative to the ontology approach, object-oriented approaches could also be considered. In any case, this approach complements ontology or object-oriented approaches.

Another disadvantage of the approach relative to standard numerical embedding is that, for list and set elements, the distance computation time is quadratic or cubic rather than linear, as in a Euclidean distance computation after a numerical embedding. However, standard numerical embedding requires a high-dimensional space, whereas in the present setup, we remain in low dimensions. In this context, distance computation using the Levenshtein or Hungarian algorithms is inherently challenging to parallelize, and specialized processors such as GPUs appear unable to accelerate these non-trivial computations.

6.3. Relation with inner brain and semiotics representation of symbols

Natural language is neither the single factor that explains the cognitive singularity, nor the more plausible vehicle for inner brain symbol manipulation [19], while the languages of thought hypothesis [5], is pertinent for instance to encounter for the mental representation of geometric shapes [56]. In contrast, [19] proposes that "humans possess multiple internal languages of thought to encode structures in various domains, and show that such symbolic representations rely on cortical circuits distinct from classical language areas". In various tasks of elementary shape or sequence perception, minimum description length in the proposed inner brain languages captures human behavior and brain activity, whereas simpler non-symbolic models capture non-human primate data. The formalism developed in this paper could be a candidate for implementing such languages of thought, with a direct numerical anchoring of inner-brain symbolic representations based on the proposed metric.

At a more abstract level, the semiotic approach, as reviewed in [18], may help structure such languages of thought by considering a notion of "sign" that elucidates the emergence of symbolic representation within a biological system interacting with its environment. This means that we attempt to consider internal cognitive representations, rather than the "external" symbols typically considered in the usual semiotics. Furthermore, in semiotics, there is no notion of hierarchy. In contrast, the fact that a sign is an Icon and Index, or a Symbol, depends on the type of representational relation such a sign bears to its object, or more precisely, to its ground [61]. We also introduce the idea that it could be organized in a hierarchy:

- An Icon has a physical resemblance to the thing being represented. Within the brain, we could consider that such a sign is directly constructed from sensorimotor features, with features such as color or smell providing an element of likeness to the described object.
- An Index meaning is built on a link between what is represented and concrete sensorimotor attributes. When looking at the proposed examples (e.g., a weathercock indexing the wind direction and strength), it appears that the concrete sensorimotor attributes correspond to an Icon (a weathercock Icon in the given example). We thus propose that an Index be built on concrete relationships among Icons representing described objects.
- A Symbol has no resemblance between the signifier and what is signified (here available through sensorymotor cues), and it must be learned (culturally learned, considering usual semiotics instantiating). They are thus built on abstract general relationships between concrete concepts, Index, or sensorimotor features, Icons, with a qualitative break-up regarding concrete object features.

Such an emergence of more abstract signs from sensorimotor icons, as pointed out by [51], who details such a process, is the dual concept of grounding and is coined as “ungrounding”, and is well accepted in many modeling works and bio-inspired artificial cognitive architectures, as recently reviewed by [39]. The research question here is whether such an emergence could correspond to different types of signs as studied in semiotics. Again, the proposed formalism could be a candidate for representing internal signs in such languages of thought and for exploring how to process them.

A step further, at the computational neuroscience level, [2] describes the systemic functional aspect of the brain processes as interactions between three kinds of memories:

- sensorimotor distributed, in a nutshell, in the parietal and medial cortex,
- episodic memory in relation to the hippocampus, and
- semantic memory in the prefrontal and frontal cortex.

One key aspect is that, on the one hand, the hippocampus does not store object features but instead indexes sensorimotor areas that represent those features, i.e., icons, in semiotic terms. It also stores temporal and spatial information and the relations among features, and it feeds into semantic memory, which might correspond to an Index. The brain’s semantic memory is also described by the author, like in semiotics, as abstract general relationships, including action rules. It also corresponds to algorithmic cognitive architectures, such as the [48] architecture for creative open-ended problem solving. Her architecture includes a feature level, as other frameworks presented previously. At the same time, ungrounding is organized in two layers: the lower layer concerns the relation between allowing the emergence of concepts, and the upper layer is more “structural” and comprises problem-solving rules, as attributed to the brain’s semantic memory. However, beyond the need for a two-layer hierarchy above the sensorimotor layer and similar representational properties, the mapping is not obviously one-to-one; it varies with the modeling aspects and the target application architecture. To what extent could these brain architecture map onto such semiotics inspired taxonomy could be an interesting issue, and using the proposed formalism a way to very it a falsifiable level.

We summarize these putative ideas in Figure 3 and propose applying our formalism to address these questions.

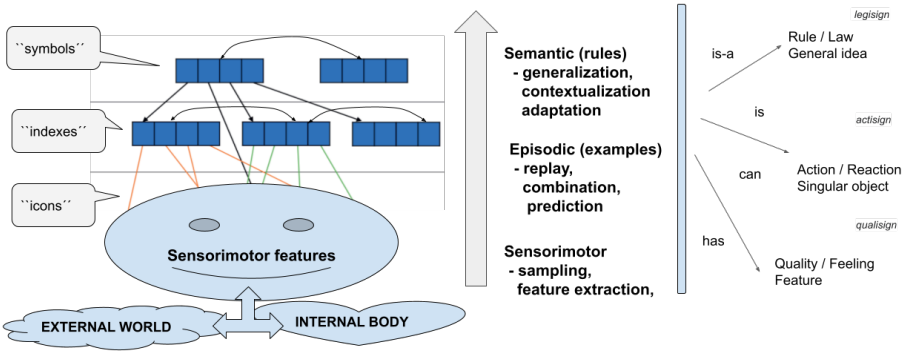


Fig. 16. Summarizing a few multi-disciplinary assumptions that may enlighten what is actually considered regarding inner-brain symbolic representation.

6.4. On the biological plausibility of the proposed knowledge representation

Although beyond the scope of this paper, it is worth noting that the biologically plausible nature of such knowledge has been considered in other studies. Symbolic reasoning occurring in the cortical-thematic loops via the basal ganglia has been successfully modeled using spiking neurons within Vector Symbolic Approaches (VSA), as introduced by [60]. In contrast, [44] has recently described how ontology representation can be encoded in such a way that reasoning entailment rules correspond to standard biological neuronal processing. Furthermore, in [17], it is

shown that such a Semantic Pointer Architecture scales appropriately to address large-scale problems and enables encoding of data structures as defined here. These developments include creativity tests simulation, as studied by [37], who have proposed a spiking network model to account for the Remote Associates Test simulation, storing the employed representations and reproducing human prediction, or well-posed problem solving, such as the Tower of Hanoi task simulated by [59].

Such biologically plausible numerical grounding of this data structure is fundamentally different from the previously proposed sub-symbolic numerical grounding, because VSA approaches operate on a high-dimensional, compact (typically hyperspherical) space of randomly drawn vectors. In contrast, basic symbolic operations are represented by abstract algebraic operators, implemented as distributed operators simulated in biological networks. A precise development of VSA along the lines proposed here constitutes a research project in its own right. Still, the basic ingredients are present, and recent work has described how all data structures used in the present developments can be implemented in VSA [40]. Thanks to this ongoing work, we have all the ingredients for a biologically plausible implementation of what we call here "symboling" mechanisms.

6.5. Perspectives

To extend this preliminary work, at the concrete programming level, we have introduced several functionalities, for web interface, procedural interface, knowledge graphs, and ontology reasoners, explicitized in appendix B. This is presented to provide an overview of potential tools that could be employed to extend the present work, as well as their feasibility of implementation. It also includes theoretical remarks on the relationship to other formalisms.

We now discuss additional perspectives across several application domains.

6.5.1. Application to complex problem solving

We have defined problem-solving tasks at a geometric level, considering a state-space location and a final (unique or alternative) state, and finding a path from the former to the latter while satisfying path constraints. This definition is computationally efficient, even for a complex symbolic state space, because we have developed a fully formalized problem-solving algorithm in which each element is precisely defined at a metaphorical level. This may be applied in several computational domains, including robotic trajectory generation and optimization, transportation theory, and operations research. We are interested in ill-posed problems, which means that estimating the initial state, selecting final goal parameters, identifying the relevant region of the state space, and generating the trajectory must be addressed during the problem-solving task and are integral to it. This means that the conceptual representation must be adapted during task execution, thereby being completed and corrected, as previously explored in [52].

6.5.2. Application to creative problem solving

Creative problem solving requires divergent thinking as analyzed in [54] and formalized in [4]. In brief, divergent thinking requires not only interpolation but also extrapolation of new data structures from existing ones. Let us illustrate these opportunities by considering three non-exclusive examples, as illustrated in Fig 17:

- *Projective divergent extrapolation:* Given a present state represented as a data structure, we may wish to extrapolate another state, only constrained by some requirement. For instance, we may want to invent a "penguinemu" (a penguin morphing towards an emu) for which the only constraint is that it diverges from the usual emu's features. Building on the notion of schema proposed above, used here to specify the target requirements, and the notion of a projector, we have an effective mechanism to generate an extrapolated, unprecedented data structure by projecting the prototype onto the target schema constraints.
- *Sequential extrapolation:* Let us consider two data structures and the editing sequence defining a path from the former to the latter, for instance, the yesterday state and the today state, whatever this means, re-applying the editing sequence on the today state allows us to extrapolate what could be the tomorrow state, under the assumption that the evolution will be the same. We could also add randomness to explore alternatives or, as in the previous example, introduce constraints to conform to any requirement.
- *Reasoning by analogy:* Following the definition of analogy reasoning proposed by [33], a reasoning of the form "Robin is to Batman what Sancho is to Don Quixote" can be formalized using the editing sequence from Robin to Batman in the source context and reapplying for Sancho in the target context in order to generate by analogy features that could apply to Don Quixote. The mapping from Robin to Sancho, from the source to the target domain, forms a commutative diagram with the source and target relations. This mechanism is

iterative, as shown in [33], which considers the ontology. Here, we propose implementing it at the geometric level.

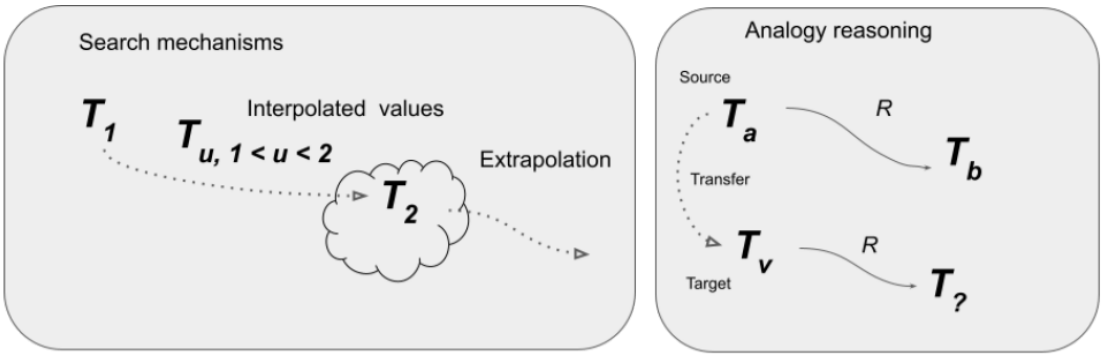
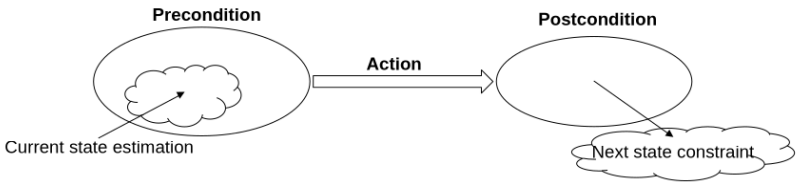


Fig. 17. A schematic representation of two kinds of generative processes. Left: search, by extrapolation in the mirror of an interpolation between two structures. Right: reasoning by analogy, as formalized in [33].

6.5.3. Application to higher-level resources specification

These mechanisms also allow us to define not only "objects" but also higher-level resources such as rules, as schematized in Figure 18, where the pre-condition of application corresponds to the fact the current state is compliant with respect to a given type, as defined in this contribution, while the post-condition corresponds to an editing sequence of the current state. A step further, we can not only define positively defined knowledge but also add a property to defined uncertain, approximate knowledge, with a particular belief level, regarding what is stated, as formalized in appendix A.3.




```
my-rule: {
 pre_condition: schema
 post_condition: editing-sequence
 action: ../..
}
```

Fig. 18. An example of higher-level specification, to formalize rule-based behavior, as discussed in [2].

7. Conclusion

We thus have been able to directly define on symbolic data structures, thanks to a parameterized editing distance, and several numerical tools, allowing us to apply potent algorithms, as summarized in Figure 19, showing what is to be defined at the design level (Figure 19.A) and what is derived thanks to this (Figure 19.B). Moreover, as

schematized in Figure 19, region inclusion yields a class taxonomy. At the same time, state value attributes and state value properties can be interpreted as a scalar-field on their domain × range on their Cartesian product: not addressed here, this is an interesting perspective of this work, since it allows to link, thanks to the geometric embedding, these symbols to basic ontology concepts, as discussed for instance in [39].

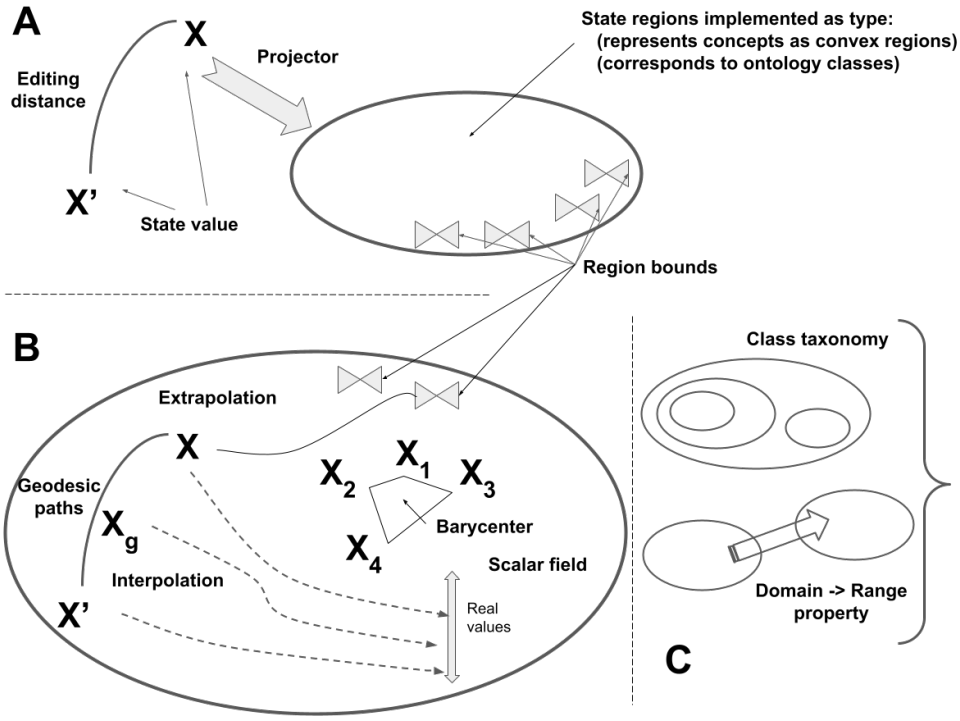


Fig. 19. Summarizing the different "symboling" tools:
A: Tools defined at the design level: the notion of a region corresponding to a type or concept, equipped with a projector, and equipped with a metric using an editing distance between the different state values, while bounds are to be specified.
B: Tools derived thanks to the previous specifications: geodesic paths, value's interpolation and extrapolation, barycenter computation, and scalar-field allowing to manage numeric criterion, reward function, or trajectory potential.
C: The notion of region is in one-to-one correspondence with class taxonomy, using the inclusion relation, and state value attributes and state value properties can be defined within this formalism.

Altogether, our design choice is to consider that any data structure is defined by a simple data structure with stated, calculated, and deduced features, and mapped onto a multi-dimensional parametric space, on which we can determine distance and geodesic between data structures, projection onto a data structure subset, and manipulation via a coordinate system. Nothing "new" here: we voluntarily sit on existing well-established formalism, thus directly benefiting from sophisticated specification paradigms and entailment algorithms. Our add-on is at the *design choices* level, making explicit how these different aspects of the chosen tools can interface and inter-operate, including in learning algorithms, as developed in [41] for a preliminary version of the present framework.

7.1. Acknowledgments and contributions

The NAI journal reviewers are especially and gratefully acknowledged for their thorough reviews, which provided valuable feedback that enabled us to improve this paper substantially.

Frédéric Alexandre is widely acknowledged for his seminal ideas that underpin this work, his shared expertise in its computational neuroscience aspects, and his technical advice throughout the work.

Margarida Romero is gratefully acknowledged for scientific discussions at the conceptual and phenomenological level regarding creative problem-solving and human learning.

Chloé Mercier and Axel Palaude have made substantial contributions to the core scientific ideas of this work, co-formulated theoretical aspects for more than 30% each, and contributed to the paper's construction and writing. Thierry Viéville has synthesized these elements and written the first draft of the paper.

The morphing experiment illustrated in subsection 5.1 was developed by Paul Bernard, Benjamin Hate, and Morgane Laval during their engineering curriculum, under the supervision of Chloé Mercier.

The climbing experiment illustration in subsection 5.2 was developed by Thibaut Lanier, Léo-Paul Mazière, and Priscilla Tissot during their engineering curriculum, under the supervision of Axel Palaude.

The survival experiment illustration in subsection 5.3 has been developed by Waris Radji, Corentin Léger, and Lucas Bardisbanian during their engineering curriculum, under the supervision of Chloé Mercier.

References

[1] H. Abbes and F. Gargouri, Modular Ontologies Composition: Levenshtein-Distance-Based Concepts Structure Comparison, *International Journal of Information Technology and Web Engineering* **13**(4) (2018), 35–60.

[2] F. Alexandre, A global framework for a systemic view of brain modeling, *Brain Informatics* **8**(1) (2021), 3.

[3] F. Alexandre, C. Mercier, A. Palaude, M. Romero and T. Vieville, Modeling Creative Problem-Solving tasks from a computational and neuroeducational approach, 2024, submitted.

[4] F. Alexandre, C. Mercier, A. Palaude and M. Romero, Modeling Creative Problem-Solving Tasks from a Computational and Neuroeducational Approach, report, Inria & Labri, Univ. Bordeaux, 2024, Pages: 27. https://inria.hal.science/hal-04691371.

[5] M. Aydede, The Language of Thought Hypothesis, in: *Stanford Encyclopedia of Philosophy*, E. Zalta, ed., 2010.

[6] S. Badreddine, A.d. Garcez, L. Serafini and M. Spranger, Logic Tensor Networks, 2021, arXiv: 2012.13635. http://arxiv.org/abs/2012.13635.

[7] P. Bernard, B. Hate and M. Laval, Symboling : utiliser des structures symboliques dotées d'une métrique, Research Report, RR-9499, Inria & Labri, Univ. Bordeaux, 2023, Issue: RR-9499. https://inria.hal.science/hal-04006574.

[8] R. Betzel and D. Bassett, Multi-scale brain networks, *NeuroImage* **160** (2016).

[9] P. Bille, A survey on tree edit distance and related problems, *Theoretical Computer Science* **337**(1) (2005), 217–239. https://www.sciencedirect.com/science/article/pii/S0304397505000174.

[10] D.B. Blumenthal, New Techniques for Graph Edit Distance Computation, PhD thesis, Faculty of Computer Science, Free University of Bozen-Bolzano, 2019, arXiv: 1908.00265. http://arxiv.org/abs/1908.00265.

[11] A. Boumaza and B. Scherrer, Optimal control subsumes harmonic control, in: *Proceedings 2007 IEEE International Conference on Robotics and Automation*, IEEE, 2007, pp. 2841–2846, ISSN: 1050-4729. https://hal.inria.fr/inria-00170185.

[12] M. Buehren, Functions for the rectangular assignment problem, 2023. https://www.mathworks.com/matlabcentral/fileexchange/6543-functions-for-the-rectangular-assignment-problem.

[13] C. Burges, Dimension Reduction: A Guided Tour, *Foundations and Trends in Machine Learning* **2** (2010).

[14] N. Burkart and M.F. Huber, A Survey on the Explainability of Supervised Machine Learning, *Journal of Artificial Intelligence Research* **70** (2021), 245–317. https://jair.org/index.php/jair/article/view/12228.

[15] S.T. Cao, L.A. Nguyen and A. Szałas, The Web Ontology Rule Language OWL 2 RL + and Its Extensions, in: *Transactions on Computational Intelligence XIII*, N.-T. Nguyen and H.A. Le-Thi, eds, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2014, pp. 152–175. ISBN 978-3-642-54455-2.

[16] I. Chraibi Kaadoud, A. Bennetot, B. Mawhin, V. Charisi and N. Díaz-Rodríguez, Explaining Aha! moments in artificial agents through IKE-XAI: Implicit Knowledge Extraction for eXplainable AI, *Neural Networks* **155** (2022), 95–118. https://www.sciencedirect.com/science/article/pii/S0893608022003021.

[17] E. Crawford, M. Gingerich and C. Eliasmith, Biologically Plausible, Human-Scale Knowledge Representation, *Cognitive Science* **40**(4) (2016), 782–821.

[18] T. De Villiers, Why Peirce matters: the symbol in Deacon's Symbolic Species, *Language Sciences* **29**(1) (2007), 88–108. https://philarchive.org/rec/DEVWPM-4.

[19] S. Dehaene, F. Al Roumi, Y. Lakretz, S. Planton and M. Sablé-Meyer, Symbols and mental programs: a hypothesis about human singularity, *Trends in Cognitive Sciences* **26**(9) (2022), 751–766. https://www.sciencedirect.com/science/article/pii/S1364661322001413.

[20] T. Denœux, D. Dubois and H. Prade, Representations of Uncertainty in AI: Beyond Probability and Possibility, in: *A Guided Tour of Artificial Intelligence Research: Volume I: Knowledge Representation, Reasoning and Learning*, P. Marquis, O. Papini and H. Prade, eds, Springer International Publishing, Cham, 2020, pp. 119–150. ISBN 978-3-030-06164-7.

[21] T. Denœux, D. Dubois and H. Prade, Representations of Uncertainty in AI: Probability and Possibility, in: *A Guided Tour of Artificial Intelligence Research: Volume I: Knowledge Representation, Reasoning and Learning*, P. Marquis, O. Papini and H. Prade, eds, Springer International Publishing, Cham, 2020, pp. 69–117. ISBN 978-3-030-06164-7.

[22] R. Desislavov, F. Martínez-Plumed and J. Hernández-Orallo, Compute and Energy Consumption Trends in Deep Learning Inference, *Sustainable Computing: Informatics and Systems* **38** (2023), 100857, arXiv:2109.05472 [cs]. http://arxiv.org/abs/2109.05472.

[23] M. Dojchinovski, J. Forberg, J. Frey, M. Hofer, D. Streitmatter and K. Yankov, The DBpedia Technology Tutorial, in: *Proceedings of the Workshops and Tutorials held at LDK 2021*, S. Carvalho, R.R. Souza, E. Daga, J. Gracia, B. Kabashi, I. Kernerman, A. Meroño-Peñuela, V. Presutti, S. Tonelli, R. Troncy, M.v. Erp and S. Žitnik, eds, CEUR Workshop Proceedings, Vol. 3064, CEUR, Zaragoza, Spain, 2021, pp. 221–228, ISSN: 1613-0073. https://ceur-ws.org/Vol-3064/#DBpedia-Technology-tutorial.

[24] H. Eichenbaum, Memory: Organization and Control, *Annual Review of Psychology* **68**(1) (2017), 19–45.

[25] C. Eliasmith, *How to Build a Brain: A Neural Architecture for Biological Cognition*, OUP USA, 2013, Google-Books-ID: BK0YRJPmuzgC. ISBN 978-0-19-979454-6.

[26] D. Fensel, E. Motta, S. Decker and Z. Zdrahal, Using ontologies for defining tasks, problem-solving methods and their mappings, in: *Knowledge Acquisition, Modeling and Management*, Vol. 1319, J.G. Carbonell, J. Siekmann, G. Goos, J. Hartmanis, J. van Leeuwen, E. Plaza and R. Benjamins, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 113–128, Series Title: Lecture Notes in Computer Science. ISBN 978-3-540-63592-5 978-3-540-69606-3.

[27] B. Fischer, Modal Epistemology: Knowledge of Possibility & Necessity, 2018. https://1000wordphilosophy.com/2018/02/13/modal-epistemology/.

[28] A.d. Garcez and L.C. Lamb, Neurosymbolic AI: the 3rd wave, *Artificial Intelligence Review* **0** (2023).

[29] E. Gilbert and D. Johnson, Distance functions and their application to robot path planning in the presence of obstacles, *IEEE Journal on Robotics and Automation* **1**(1) (1985), 21–30. https://ieeexplore.ieee.org/document/1087003.

[30] P. Gleeson, M. Cantarelli, B. Marin, A. Quintana, M. Earnshaw, S. Sadeh, E. Piasini, J. Birgiolas, R.C. Cannon, N.A. Cayco-Gajic, S. Crook, A.P. Davison, S. Dura-Bernal, A. Ecker, M.L. Hines, G. Idili, F. Lanore, S.D. Larson, W.W. Lytton, A. Majumdar, R.A. McDougal, S. Sivagnanam, S. Solinas, R. Stanislovas, S.J.v. Albada, W.v. Geit and R.A. Silver, Open Source Brain: A Collaborative Resource for Visualizing, Analyzing, Simulating, and Developing Standardized Models of Neurons and Circuits, *Neuron* **103**(3) (2019), 395–411.e5, Publisher: Elsevier. https://www.cell.com/neuron/abstract/S0896-6273(19)30444-1.

[31] A. Glennerster, Computational theories of vision, *Current biology: CB* **12**(20) (2002), R682–685.

[32] P. Gärdenfors, Conceptual Spaces as a Framework for Knowledge Representation, *Mind and Matter* **2** (2004), 9–27.

[33] J. Han, F. Shi, L. Chen and P.R.N. Childs, A computational tool for creative idea generation based on analogical reasoning and ontology, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **32**(4) (2018), 462–477. https://www.cambridge.org/core/product/identifier/S0890060418000082/type/journal_article.

[34] V.G. Hardcastle and K. Hardcastle, Marr's Levels Revisited: Understanding How Brains Break, *Topics in Cognitive Science* **7**(2) (2015), 259–273, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/tops.12130.

[35] S. Harnad, The symbol grounding problem, *Physica D: Nonlinear Phenomena* **42**(1) (1990), 335–346. https://www.sciencedirect.com/science/article/pii/0167278990900876.

[36] P. Hohenecker and T. Lukasiewicz, Ontology Reasoning with Deep Neural Networks, *Journal of Artificial Intelligence Research* **68** (2020), 503–540. https://jair.org/index.php/jair/article/view/11661.

[37] I. Kajić, J. Gosmann, T.C. Stewart, T. Wennekers and C. Eliasmith, A Spiking Neuron Model of Word Associations for the Remote Associates Test, *Frontiers in Psychology* **8** (2017), 99.

[38] J.L. McClelland and T.T. Rogers, The parallel distributed processing approach to semantic cognition, *Nature Reviews Neuroscience* **4**(4) (2003), 310–322. http://www.nature.com/articles/nrn1076.

[39] C. Mercier, Modeling cognitive processes within a creative problem-solving task: from symbolic to neuro-symbolic approaches in computational learning sciences, PhD thesis, U. Bordeaux, 2024, Computational Learning Sciences, Creative Problem Solving, Knowledge Representation and Reasoning, Reinforcement Learning, Vector Symbolic Architectures..

[40] C. Mercier and T. Vieville, Algorithmic ersatz for VSA: Macroscopic simulation of Vector Symbolic Architecture, *Neurosymbolic Artificial Intelligence* **submitted** (2025).

[41] C. Mercier, F. Alexandre and T. Viéville, Reinforcement Symbolic Learning, in: *Artificial Neural Networks and Machine Learning – ICANN 2021*, I. Farkaš, P. Masulli, S. Otte and S. Wermter, eds, Lecture Notes in Computer Science, Vol. 12894, Springer International Publishing, Bratislava, Slovakia / Virtual, 2021, pp. 608–612. ISBN 978-3-030-86380-7. https://inria.hal.science/hal-03327706.

[42] C. Mercier, F. Alexandre and T. Viéville, Ontology as manifold: towards symbolic and numerical artificial embedding, Lulea, Sweden / Virtual, 2022, Presenters: _:n230. https://inria.hal.science/hal-03550354.

[43] C. Mercier, L. Roux, M. Romero, F. Alexandre and T. Viéville, Formalizing Problem Solving in Computational Thinking : an Ontology approach, in: *IEEE ICDL 2021 – International Conference on Development and Learning 2021*, Beijing, China, 2021. https://hal.inria.fr/hal-03324136.

[44] C. Mercier, H. Chateau-Laurent, F. Alexandre and T. Viéville, Ontology as neuronal-space manifold: Towards symbolic and numerical artificial embedding, in: *KRHCAI 2021 Workshop on Knowledge Representation for Hybrid & Compositional AI @ KR2021*, Hanoi, Vietnam, 2021. https://hal.inria.fr/hal-03360307.

[45] G. Navarro, A guided tour to approximate string matching, *ACM Comput. Surv.* **33**(1) (2001), 31–88.

[46] A. Newell and H.A. Simon, *Human problem solving*, Prentice-Hall, Englewood Cliffs, N.J., 1972, OCLC: 622041645.

[47] N. Noy and D.L. McGuinness, Ontology Development 101: A Guide to Creating Your First Ontology, Technical Report, Stanford University, 2001. https://protege.stanford.edu/publications/ontology_development/ontology101.pdf.

[48] A.-M. Olteţeanu, *Cognition and the Creative Machine: Cognitive AI for Creative Problem Solving*, Springer International Publishing, Cham, 2020. ISBN 978-3-030-30322-8.

[49] A. Ouangraoua and P. Ferraro, A Constrained Edit Distance Algorithm Between Semi-ordered Trees, *Theoretical Computer Science* **410**(8–10) (2009), 837–846, Publisher: Elsevier. https://hal.archives-ouvertes.fr/hal-00350113.

[50] D. Pandove, R. Rani and S. Goel, Local graph based correlation clustering, *Knowledge-Based Systems* **138** (2017), 155–175. https://www.sciencedirect.com/science/article/pii/S0950705117304537.
[51] J. Raczaszek-Leonardi and T.W. Deacon, Ungrounding symbols in language development: implications for modeling emergent symbolic communication in artificial systems, in: *2018 Joint IEEE 8th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, 2018, pp. 232–237, ISSN: 2161-9484.
[52] W. Radji, C. Léger and L. Bardisbanian, Early Empirical Results on Reinforcement Symbolic Learning, Research Report, RR-9509, Inria & Labri, Univ. Bordeaux, ENSEIRB Bordeaux INP, Bordeaux, France, 2023. https://inria.hal.science/hal-04103795.
[53] M. Romero, D. David and B. Lille, CreaCube, a Playful Activity with Modular Robotics, in: *Games and Learning Alliance*, Vol. 11385, M. Gentile, M. Allegra and H. Söbke, eds, Springer International Publishing, Cham, 2019, pp. 397–405, Series Title: Lecture Notes in Computer Science. ISBN 978-3-030-11548-7.
[54] M. Romero, A. Palaude, C. Mercier and F. Alexandre, Deciphering Cognitive Processes in Creative Problem Solving: A Computational Approach, report, Inria & Labri, Université de Bordeaux, 2024, Pages: 26. https://inria.hal.science/hal-04691368.
[55] A.L. Rusawuk, Possibility and Necessity: An Introduction to Modality, 2018. https://1000wordphilosophy.com/2018/12/08/possibility-and-necessity-an-introduction-to-modality/.
[56] M. Sablé-Meyer, K. Ellis, J. Tenenbaum and S. Dehaene, A language of thought for the mental representation of geometric shapes, *Cognitive Psychology* **139** (2022), 101527.
[57] R. Schaeffer, M. Khona and I.R. Fiete, No Free Lunch from Deep Learning in Neuroscience: A Case Study through Models of the Entorhinal-Hippocampal Circuit, in: *Proceedings NeurIPS 2022*, 2022. https://openreview.net/forum?id=syU-XvinTI1.
[58] L. Smith, The development of modal understanding: Piaget's possibility and necessity, *New Ideas in Psychology* **12**(1) (1994), 73–87. https://www.sciencedirect.com/science/article/pii/0732118X94900590.
[59] T. Stewart and C. Eliasmith, Neural Cognitive Modelling: A Biologically Constrained Spiking Neuron Model of the Tower of Hanoi Task, *Proceedings of the Annual Meeting of the Cognitive Science Society* **33**(33) (2011). https://escholarship.org/uc/item/6kv930kg.
[60] T.C. Stewart, X. Choo and C. Eliasmith, Symbolic Reasoning in Spiking Neurons: A Model of the Cortex/Basal Ganglia/Thalamus Loop, *Proceedings of the Annual Meeting of the Cognitive Science Society* **32** (2010), 7.
[61] C. Sánchez-Ovcharov and M. Suárez, Peirce's Pragmatism, Semiotics, and Physical Representation, *European Journal of Pragmatism and American Philosophy* **XVI**(1) (2024), Publisher: Associazione Pragma. https://journals.openedition.org/ejpap/3817.
[62] M. Taddeo and L. Floridi, Solving the Symbol Grounding Problem: A Critical Review of Fifteen Years of Research, *Journal of Experimental and Theoretical Artificial Intelligence* **17** (2005).
[63] A. Tettamanzi, C.F. Zucker and F. Gandon, Possibilistic testing of OWL axioms against RDF data, *International Journal of Approximate Reasoning* **91** (2017), 114–130. https://hal.inria.fr/hal-01591001.
[64] T. Vallaeys, Généraliser les possibilités-nécessités pour l'apprentissage profond, Research Report, RR-9422, Inria, 2021, Issue: RR-9422. https://inria.hal.science/hal-03338721.
[65] S.L. Vasileiou, W. Yeoh, T.C. Son, A. Kumar, M. Cashmore and D. Magazzeni, A Logic-Based Explanation Generation Framework for Classical and Hybrid Planning Problems, *Journal of Artificial Intelligence Research* **73** (2022), 1473–1534. https://jair.org/index.php/jair/article/view/13431.
[66] T. Viéville, An improved biologically plausible trajectory generator, Research report, RR-4539, Inria, 2002. https://hal.inria.fr/inria-00072049.
[67] T. Viéville and C. Mercier, Representation of belief in relation to randomness, Research Report, RR-9493, Inria & Labri, Univ. Bordeaux, 2022, Issue: RR-9493. https://inria.hal.science/hal-03886219.
[68] T. Viéville, D. Lingrand and F. Gaspard, Implementing a multi-model estimation method, *International Journal of Computer Vision* **44** (2001), 41–64. https://hal.inria.fr/inria-00000172.

Appendix A. Symbolic data implementation

In this section, we recapitulate for each implemented data type the different required ingredients: syntactic and semantic projections, distance and geodesic path, and default comparison.

A.1. StringType specification

– **Syntactic projection**
  - If the value is not a string but a structured value, its string representation is taken into account.
  - If the string syntax is incorrect concerning a lexical constraint or if the string length is not in the optional length bounds, a message is issued, but the string is unchanged.
– **Semantic projection**
  - Any value has a string representation.

– **Distance value**
  - The distance is computed as a usual edit distance on the char list of the string.
– **Geodesic path**
  - If considered as an atomic value, the geodesic is of length 1 if the string is equal, and 2 otherwise.
  - If not atomic, the geodesic is computed from the edit distance, as for a ListType.
– **Comparison**
  - Returns 0 if each character pair compares equal and both strings have the same length.
  - Returns <0 if either the value of the first char that does not match is lower in the left-hand side string, or all compared chars match but the left-hand side string is shorter.
  - Returns >0 if either the value of the first char that does not match is greater in the left-hand side string, or all compared chars match but the left-hand side string is longer.

A.2. NumericType specification

– **Syntactic projection**
  - Returns *NaN* (i.e. the "Not a Number" meta-value) if not a parsable number.
  - Returns the zero default value if an empty value.
– **Semantic projection**
  - If the value is higher than the maximal value or lower than the minimal value, it is projected on the related bound.
  - If the precision is defined, the value is projected to the closest $min + k\ precision$ value, k being an integer.
– **Distance value**
  -The distance between two values is calculated as a weighted value $d(lhs, rhs) = |lhs - rhs|/step$.
  - It is $INFINITY$ if not both objects are numeric.
– **Geodesic path**
  -The geodesic is assumed to be atomic: of length 1 if values are equal and 2 if different.
– **Comparison**
  - Returns either $lhs - rhs$ or 0 if the precision is defined and |lhs-rhs| is below the precision.

Numerical quantity related to a sensory input or an algorithm numerical value, corresponds to a bounded value (between minimal and maximal bounds), up to a given precision threshold (above which two values may differs and below which two values are indistinguishable, being equal or not), an approximate neighborhood sampling size or "step" (below which two distinct values are in the same local area), a default value (used in initialization, or to avoid undefined value), expressed in a given unit (for instance, second, meter, etc), if any, as schematized in Figure 20.

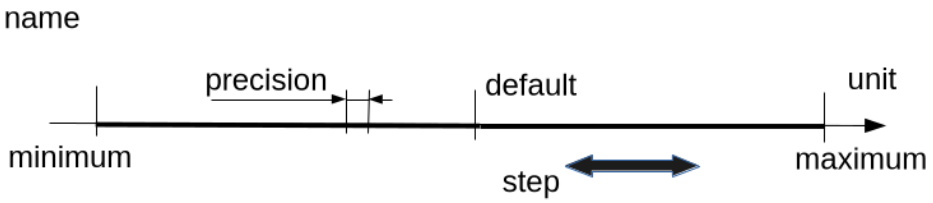


Fig. 20. Specification of a numerical value. Metadata includes a name, a unit, a default value, bounds (minimum and maximum), a precision under which two values are not distinguishable, and a step value corresponding to a neighborhood size used to cover the value range.

Such specification is essential for the proper manipulation of quantitative information. In particular, the values can be normalized, meaning mapped to the $[-1, 1]$ interval or to a finite set of relevant values. One consequence is that algorithmic precision thresholds can be deduced (often using first-order approximations), spurious values can

be detected, and numerical conditioning of algorithms is enhanced, as proposed and applied in [68], which discusses this specification. At the computational specification level, these parameters define a subtype of the usual numerical types, thereby providing a more precise definition of the related code.

This concerns both external numerical sensory data and internal quantitative data, having always bounds and precision. It is evident that any quantitative measure is bounded; for instance, physical velocity magnitude stands between 0, for a still object, and the speed of light. It is also always given with a specified precision; for instance, a localization in an image is shown to within one pixel, or a school ruler to within 1 mm. The key point is that it is helpful to make this obvious fact explicit at the specification level, rather than using it implicitly when required, to enable its general use in algorithms. For instance, an iterative estimation algorithm, thanks to this specification, can be easily tuned to achieve the required precision. A step further, symbolically coded data can be sampled continuously. For instance, a vector font drawn on a canvas is sampled at pixel precision.

The positive sampling step, used to define a local neighborhood size, is employed to weight distance calculations and to adequately sample the data space. The underlying idea is that the state space is locally convex, so that in a given neighborhood, local search for an optimum yields the optimal local value. This idea has been implemented in the stepsolver variational solver, including optimizer and controller.

This specification induces a pseudo-metric:

$$d(x,x') = \frac{|x-x'|}{step}, |x-x'| > precision \Rightarrow x \neq x',$$

in words, the distance is weighted by the *step*, which is the neighborhood's approximate size. At the same time, if two values differ by a quantity below the precision threshold, they are indistinguishable (thus either equal or not), so that we can decide if two values are different but not decide about their equality, as usual in such cases. This a deterministic counterpart of statistic hypothesis test, with $\mathcal{H}_1$ versus $\mathcal{H}_0$ hypothesizes.

A.3. ModalType specification

- **Syntactic projection**
  - Returns *NaN* if not a parsable number or a predefined string.
  - Returns 0 if an empty value.
  - The values "true" (numerically 1), "false" (numerically -1), and "unknown" or '?' (numerically 0) are understood in case-insensitive mode.
- **Semantic projection**
  - If the value is higher than 1, it is set to true.
  - If the value is lower than -1, it is set to false.
  - If the precision is defined, the value is projected to the closest $min + k\ precision$ value, k being an integer.
  - If in trinary mode, values are projected onto the closest {-1, 0, 1} value.
  - If in binary mode, values are projected onto the closest {-1, 1} value.
- **Distance value**, **Geodesic path**, **Comparison**
  - These calculations are delegated to the NumericType since ModalType is a subtype of it.

The Boolean notion of being either false or true is generalized here as a numeric representation of partial knowledge, considering a value $\tau \in [-1,1]$, as illustrated in Figure 21. The interpretation is that what is "not so false" is partially possible but not necessary, and what is "partially true" is entirely possible but partially necessary. Such a formulation qualitatively corresponds to human appreciation of the degree of belief in a fact.

The true value corresponds to 1 (entirely possible and fully necessary), the false value to -1 (neither possible nor necessary, thus impossible), and the unknown value to 0, which corresponds to an entirely possible but absolutely not necessary value. In between, negative values correspond to partially possible events and positive values to partially necessary events. This representation has been designed to be compatible with the ternary Kleene logic, and is in one-to-one correspondence with respect to the possibility theory. Possibility theory is devoted to modeling incomplete information, particularly an observer's belief about a potential event and surprise after its occurrence. Please refer to [20] for a general introduction. While almost all partially known information concerns probability, human "level of truth" is subtler and concerns possibility and necessity, as formalized in possibility theory. This is

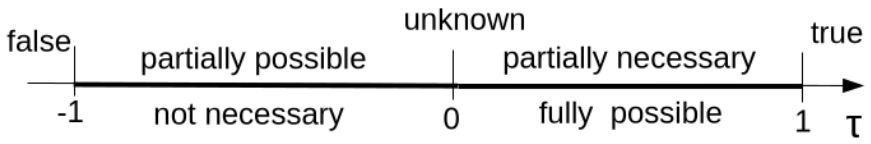


Fig. 21. Specification of a modal value of belief, this about necessity and possibility, as defined in the possibility theory.

introduced and discussed in [20] and [21]. This is linked to modal logic, modeling something as true in a "given context," as developed by [27]. Interestingly, this is also considered representative of what is modeled in educational science and philosophy, as reviewed by [55], because it corresponds to commonsense reasoning in Piaget's sense [58], taking exceptions into account, thereby involving non-monotonic reasoning. Furthermore, in symbolic artificial intelligence, specifically with respect to knowledge representation via ontology, a link has been established between the necessity/possibility dual representation and ontology [63]. This must be understood as a deterministic theory, in the sense that partial knowledge is not represented by randomness. An extension that accounts for randomness is proposed in [64, 67].

Boolean true/false values and Kleene three-value logic true/unknown/false values conjunction (`and`), disjunction (`or`), exclusive disjunction (`xor`), and element set difference generalizations correspond to `min`, `max`, and polynomial operators.

A.4. RecordType specification

- **Syntactic projection**
  - If a required field is missing, its definition is forced using a default, or if none, an empty value.
  - If a forbidden field is present, its definition is erased.
  - Each record item is syntactically projected.
- **Semantic projection**
  - Each record item is semantically projected.
- **Distance value**
  - The distance is computed as the weighted sum of each record item. The corresponding algorithm is thus obviously linear in complexity.
  - The distance is computed against the default or, if none, an empty value if one record item is missing.
- **Geodesic path**
  - The path is constructed in the record item order, scanning both record item lists simultaneously.
- **Comparison**
  - The comparison is performed in the record item order, scanning both record item lists simultaneously.
  - Returns 0 if each element pair compares equally and both lists have the same length.
  - Returns <0 if either the value of the first item that does not match is lower in the left-hand side list, or all compared items match but the left-hand side list is shorter.
  - Returns >0 if either the value of the first item that does not match is greater in the left-hand side list, or all compared items match, but the left-hand side list is longer.

A.5. EnumType specification

This type specifies an explicit enumeration of items of a given type.

- **Syntactic projection**
  - The syntactic projection is delegated to the item's type.

– **Semantic projection**
  - The semantic projection corresponds to choosing the value of minimal distance concerning the item type.
– **Distance value**
  - The minimal distance computation is delegated to the item type and applied to each value of the value set, selecting the minimal distance.
  - The complexity order of magnitude is thus linear with respect to the enumeration length, considering the item type complexity as a basic operation.
– **Geodesic path** and **Comparison**
  - These calculations are delegated to the item type.

A.6. ListType specification

– **Syntactic projection**
  - If the value is empty, it corresponds to an empty list.
  - If the value is not a list, it is encapsulated in a singleton list.
  - If the list length is not within the optional range, the list is neither truncated nor extended.
  - Each list item is syntactically projected.
– **Semantic projection**
  - Each list item is semantically projected.
– **Distance value**
  - The minimal distance is computed using the Levenshtein distance calculation applied on the list element and delegating the edition cost to the item type distance computation [45].
  - The algorithmic complexity order of magnitude is quadratic with respect to the list length, considering the item type complexity as a basic operation.
  - The editing distance can be either parameterized, defining fixed deletion, insertion, and editing costs, or parameterized by re-deriving a cost function.
– **Geodesic path**
  - The geodesic path corresponds to the Wagner-Fischer algorithm shortest path in the matrix distance, yielding the minimal distance computation.
– **Comparison**
  - Returns 0 if each element pair compares equally and both lists have the same length.
  - Returns <0 if either the value of the first unmatched item is lower in the left-hand side list, or all items match but the left-hand side list is shorter.
  - Returns >0 if either the value of the first unmatched item is greater in the left-hand side list, or all items match but the left-hand side list is longer.

A.7. SetType specification

– **Syntactic projection**
  - If the value is empty, it corresponds to an empty set.
  - If the value is not a set, it is encapsulated in a $\{value\}$ singleton.
  - If the set length is not optional, the set is neither truncated nor extended.
  - Each set item is syntactically projected.
  - To be comparable, the set is put in a canonical form:
  - Set elements are sorted according to the item type comparison.
  - Redundant elements that compare to 0 with respect to another are erased.
– **Semantic projection**
  - Each set element is semantically projected.
– **Distance value**
  - The minimal distance is computed using the Cong Ma implementation of the matrix version of the Hungarian algorithm, after [12].

- The algorithmic complexity order of magnitude is cubic with respect to the set size, considering the item type as a basic operation.
- The editing distance can be either parameterized, defining fixed deletion, insertion, and editing costs, or parameterized by re-deriving a cost function.

– **Geodesic path**
- The geodesic path is based on the Hungarian algorithm assignment map, which is scanned in the assignment list order, considering the minimal cost operation of deletion, insertion, or replacement.

– **Comparison**
- The comparison is applied to the set canonical form.
- Returns 0 if each element pair compares equally and both lists have the same length.
- Returns <0 if either the value of the first item that does not match is lower in the left-hand side list, or all compared items match but the left-hand side list is shorter.
- Returns >0 if either the value of the first item that does not match is greater in the left-hand side list, or all compared items match, but the left-hand side list is longer.

Appendix B. Data value interfacing

To extend this preliminary work, at the concrete programming level, we have introduced several functionalities for the web interface, procedural interface, knowledge graphs, and ontology reasoners. This is presented to provide an overview of potential tools that could be employed in extending the present work. At this stage, we have assessed the feasibility of the implementation.

B.1. Interfacing with the wJSON data object model

It is straightforward to notice that our scalar values map one-to-one to a JSON data structure. More precisely, at the implementation level, the data object model (DOM) is a Value which is either:

– (i) an atomic string (including a string parsable as a numeric or a boolean value) or
– (ii) a record, which is an unordered set of elements accessed by label, including list and set, because an array of syntax `[a b ...]` is equivalent to a record `{ 0: a 1: b ...}` indexed by consecutive non negative integer, and a set `{a b ...}` is equivalent to a record of the form `{a: true b:true ...}`.

This corresponds to a variant of JSON syntax in which a list is merely syntactic sugar. It is also a weak syntax, in the sense that, given an input, the parser attempts to construct the best data structure, without complaining about lexical or syntactic errors, inferring punctuation like quotes or commas, and so on. This significantly reduces editing time and allows people unfamiliar with structured syntax to write symbolic data without difficulty. This is indeed dangerous, as it may generate a spurious data structure. Still, over time, we have experimented with it, and it is pretty visible when rereading the data in a beautified format. The normalized string format is also more concise than the JSON counterpart format. Finally, any wJSON data is in one-to-one correspondence with the corresponding JSON format, thereby avoiding yet another non-standard, abstruse patch while making JSON writing easier.

This is also a semantic variant called wJSON, for which record items are sorted in insertion order by default. With this condition, all values are in one-to-one correspondence with their normalized string representation. In addition, two values are equal if and only if record items are inserted in the same order, and their string representation are equal. This also offers a pertinent absolute syntactic order between two values.

All these features are provided to facilitate web services or distributed computing, enabling data exchange in JSON or wJSON.

B.2. Interfacing with calculated or deduced feature values.

Value can not only be input as data but can also be computed, as illustrated in Figure 22. To avoid semantic ambiguity, these are treated as external mechanisms, and the resulting values are treated as new feature inputs.

In Figure 22 example, a relation relates a data structure to another (using a verb like `is-a-prey-for`) or allows to define some *class* (using the `is-a` construct). In our construction, such a feature corresponds to an

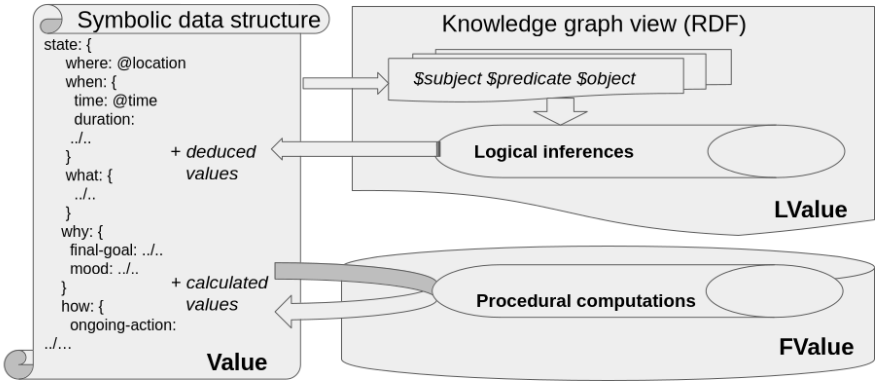


Fig. 22. Extended mechanism to compute calculated values using an external algorithmic function and deduced values using an external inference mechanism.

implicit definition of other features: The `is-a` feature implies that all features of the parent are inherited unless it is overwritten (`has-capability` in the example). In other words, some feature values, say "FValue", are calculated by algorithmic functions hooked to the data structure, and generate *calculated* features. Depending on the application needs, further *calculated* features, calculating some numerical values using a formula, could be implemented. In particular, a procedural method could interface with an external sensor or effector, for robotic applications in the broad sense. At the programming level, we have to associate a method to a data feature, which is often called a "getter" when reading and a "setter" when writing.

More precisely, we consider non-recursive imperative code constructs without side-effect, thus without global variables, made of elementary functions, sequences, and tests, and enumerators of feature set, which allows us to obtain good computational properties and a relatively simple operational semantics. One key point is that such a straight-line piece of code does not generate infinite loops and has a relatively small polynomial calculation time that can be numerically bounded. This corresponds to a common organization of code, in which straight-line code is separated from unpredictable loops or recursive sections.

Another key point is that such representation is in one-to-one correspondence with what could be specified using an ontology approach. Please refer to [43] for an introduction in this context. Very simply, individuals have a data property corresponding to a qualitative or quantitative feature property and an object property corresponding to a relation. More precisely, this corresponds to some A-box of a knowledge graph.

Thanks to this, we can add some ontology predicates, using preferentially RDFS level description, if sufficiently powerful, to define a T-box which will generate *deduced* features, say "LValue", as illustrated in Figure 23. This requires a one-to-one transformation between a hierarchical structure and a flat relational graph. One solution, called "Turtoise", is apparent and already developed, and is available here. Beyond RDFS, the OWL description language or some related derivation rules using SWRL could be added to the T-box, with the double risk of worst computation performances, and less interoperability with people not familiar with complex ontology specification.

In our actual setup, the inference rules in the T-box are defined directly in the ontology language and are not part of the data system.

When implementing LValue using existing ontology reasoners' APIs in Java or C/C++, we encountered several limitations: complex specifications, performance degradation in the interface, and, worst of all, opaque behavior. This is indeed because we attempt to use them without sufficient expertise in the tool's use. We then had the opportunity to use a helpful tool: We installed the Protégé ontology editor. This widely used ontology development environment is primarily used via a manual user interface. Using it enables us to consider a wide range of factors, obtain comprehensive feedback on the loaded items, and more. The program starts itself and uses xdotool, which simulates keyboard input and mouse activity and enables window movement and resizing, thereby replacing manual gestures with a script. The drawback is that we must run the LValue mechanism on a desktop, but given that we are still at a preliminary experimental stage, this is acceptable. This trick may have other uses.

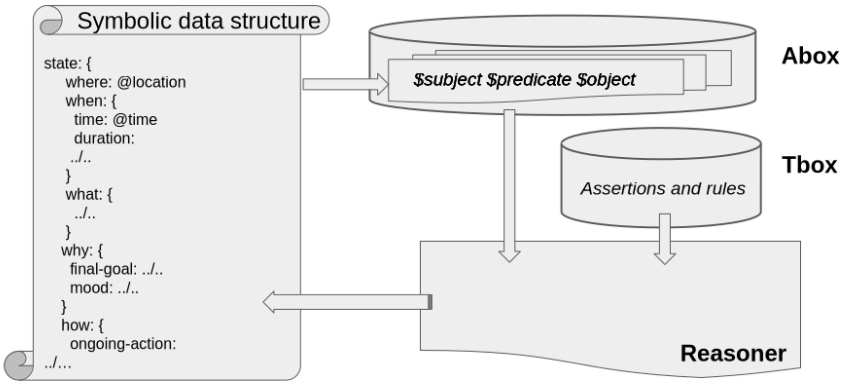


Fig. 23. Interface between the symbolic data management system and an ontology reasoner to derive feature values from inference. The input data corresponds to the A-box: the facts. The inference rules are in the T-box.