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*[6]

Abstract. Within the context of cognitive computational neuroscience modeling, and neurosymbolic artificial intelligence, with both symbolic and numerical approaches, we propose here an unusual way to relate explicit and readable data structure, with biologically plausible numeric operations. We consider a parameterized edit distance between two hierarchical symbolic data instances, thus embed data in a metric space, in such a way that we can define the notion of geodesic between two data values. We also define in this symbolic space the notion of data region, here equivalent to data type or concept, and provide projection onto such a region, so that very generic machine learning algorithms can now be applied simultaneously at both the symbolic and numerical level. This theoretical work could be interesting to other researchers in both computational neuroscience, model simulation, and artificial intelligence, to design explainable explicit ill-defined problem-solving mechanisms, because it provides a rather straightforward data representation on which generic numerical algorithms of planning or learning could be directly applied, generating explicit symbolic plans and outcomes. Even so this requires the data structure distances to be properly parameterized for a given application domain. The proposed specification is not only developed formally but implemented in detail at the programming level, with some experimental illustrations, showing the feasibility and allowing the evaluation and use of the proposed approach.

Keywords: Symbolic specification, Edit distance, Computational creativity

1. Introduction

Within the context of cognitive computational neuroscience modeling, and neurosymbolic artificial intelligence., with both symbolic and numerical approaches, we propose here an unusual way to relate explicit and readable data structure, with biologically plausible numeric operations.

To this end, in this introduction, we need to first put in context the proposed development with respect to neurosymbolic existing approaches, in order to position the originality of what is proposed here. This applies to both the system state and its related meta-data, at the neurosymbolic level. The problem-solving issue is taken as a major example. We also wish to position this work with respect to brain modeling at a symbolic level, both in terms of

[***] mailto:axel.palaude@inria.fr
[****] mailto:chloe.mercier@inria.fr
[*****] mailto:thierry.vieville@inria.fr
[6] https://neurosymbolic-ai-journal.com/content/about-neurosymbolic-artificial-intelligence
[*] 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*.

paradigm and in terms of knowledge representation. The targeted symbolic approach addresses both what is encoded in the neuronal ensembles and the modeling knowledge allowing us to explain and interpret it. The contextualization of our work, requires also to define what we precisely mean by a "symbol", and we revisit to this end what are signs and symbols, discussing the underlying concepts, and allowing us to relate our design choices, to multidisciplinary approaches considering symbolic representations. We then finally detail a modeling approach which instantiates the previous general principles, allowing us to motivate the specification developed in the sequel.

Based on this, the next section describes at the computational level, how the previous data structure ingredients can be formalized, including at the programming level, 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 geodesic. This also includes defining the notion of a concept corresponding to a region of the state space, equipped with a projector of a data point onto such a region. The notion of extrapolation, allowing exploration and search mechanisms is also designed.

The development related to the creation of a metrizable symbolic data structure embedded in a metric space, to be used in general machine learning mechanisms, requires also the specification of a scalar field, with a mechanism of interpolation and barycenter.

This is then illustrated showing a few toy demonstrations, and reporting a realistic experimentation.

A last step further, we discuss how rather generic machine learning algorithms can be applied using this formalization, in particular targeting open-ended or ill-defined complex problem-solving.

We finally discuss perspectives of this work, before a short conclusion.

Let us start putting our work in context.

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:

• In order to the paper easily readable, we provide verbal evidences in the text, while demonstrations of statements are given in footnotes with title, while only straightforward derivations are required here.

• We also provide (which is is not commonly done for scientific papers) a large set of links to allow readers from different domains to get the meaning of a non-familiar word, mainly thanks to Wikipedia. We never use such link as "scientific" reference, only as a read help.

1.1. Context: Neurosymbolic approaches

Numeric machine learning mechanism is mainly based on statistical inferences, including Bayesian inference, and has now reached outstanding performances. On the other hand, when manipulating knowledge, 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 priory knowledge is to be introduced. It has also great advantages when both interpretability (by an expert) and explainability (for end users) is a key feature, as discussed in [16], develop this dual aspects, proposing criteria of evaluation of both. These features are essential if such learning mechanisms are used to model cognitive functions. In addition, formal reasoning is obviously less costly than billions of operations of deep networks computation, as studied by [21], who compares the computation cost of both approaches, focusing on academic use for usual research teams. They also raise ecologic and ethical issues, which become a real challenge, with what is called the *artificial intelligence boom*[1].

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 [27], who also introduced a taxonomy of numeric and symbolic coupling in algorithms. The present work precisely aims at considering coupling levels where symbolic knowledge

[1] https://en.wikipedia.org/wiki/AI_boom

and rules are compiled in distributed calculation either using local mapping of symbols or more complex embedding 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 be able to introduce symbolic a priory information in numeric computations.

The interest of being able to work at both the numeric and symbolic level is well illustrated, in link with focusing on problem-solving as done here, by [62] discussing a logic-based explanation mechanism in planning, dedicated to controlled hybrid systems in human interaction. This efficient approach of the authors specifies the symbolic layer "outside" the related numeric representation, while at a more integrated level, such a plan is represented by the notion of abstract path, as developed in this paper. This work is also in link with ontology-related reasoning as, for instance, in [35] who show how symbols can be optimally encoded in the network numeric variables, before applying mechanisms of reasoning at the numerical level. In comparison, we present a dual approach, symbolic data being equipped with metric another numerical tools in our case.

A step ahead, computational neuroscience models are mainly based on numeric mechanisms, except for instance Vector Symbolic Architecture (VSA) architecture, after [24], based on Semantic Pointer Architecture (SPA). This approach introduces an intermediate algebraic abstract view of spiking neural network architectures, as performed in [17], this in order to develop biologically plausible cognitive functionalities and human knowledge representations. This also recently includes high-level symbolic representation allowing reasoning in the preliminary work of [43], regarding not only deductive but also inductive and abducting reasoning, as developed in [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 directly the symbolic data structure with the numeric ingredients needed to apply numeric algorithms, and show how general algorithms can be applied directly on the symbolic data structure by this mean. To this end, in the next section let us precisely state what is a "symbolic representation", to make explicit our design choices to represent cognitive symbolic information.

1.2. Context: Problem-solving and planning problems

A typical class of problems discussed above relates to planning problems[2]. Since [45] work, problem-solving and planning have been formalized as a trajectory problem from a initial state to a goal state, in a state space. This means avoiding "obstacle", in other words be compliant with respect to trajectory constraints, while reducing the distance to the goal. This problem statement has been recently generalized by [3] to complex open problems, with several possible goals, while the state space is only partially known.

Solutions often use a distance, including an editing distance, between two state space points, for instance, in heuristics generating step by step plans. In an early work [28], obstacle avoidance is expressed in terms of the distances between potentially colliding parts and to goal or goals, leading to the formulation of path planning problems as problem of optimal control, with variational numeric solutions. In [63], generic path planning problems have been related to a parametric harmonic potential, tractable also in high dimensions, reducing path planning to a this potential optimization, while leading to a biologically plausible solution of the hippocampus navigation skills, in either concrete locations or in abstract state space. In a nutshell: the potential is repulsive around obstacles and attractive in the direction of goals. Using some harmonic potential avoids local minima, while using parametric potential avoids the curse of dimensionality, which is the curses of usual potential methods: local minima and potential sampling in the huge state space. As for the work of [28] previously quoted, and thanks to [11] it has been shown that such harmonic control is a particular case of optimal control. In the [63] it is also equipped with a rather efficient algorithm.

With respect to such approaches, the present development will provide the setup to consider such mechanism directly using symbolic data, without the non negligible overload of embedding and the somehow arbitrary choice of a sampling.

[2]This paragraph, among other precious suggestions, is a direct suggestion of one of the reviewer, Ben J. Wright.

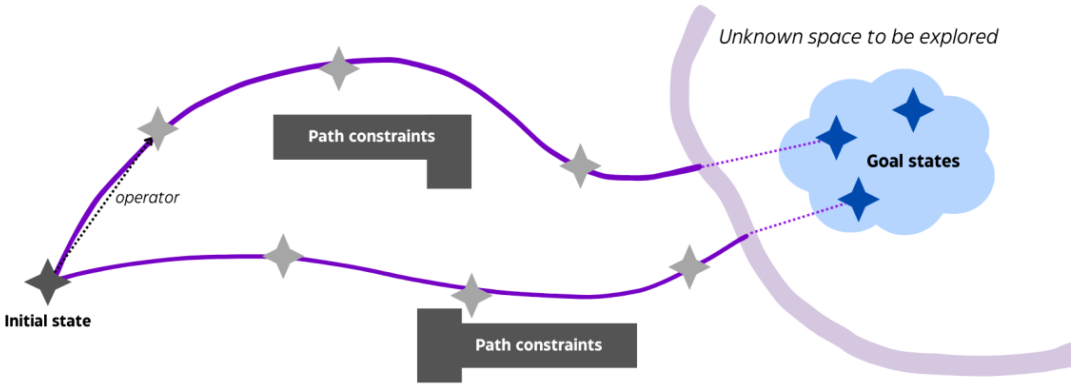


Fig. 1. Problem-solving as trajectory generation, from [39].

1.3. Context: Brain modeling at a symbolic level

Studying the brain is performed at different temporal and spatial scales, different topological network scales, which is reviewed in the overview work of [8], but also at different "modeling" scales, in the sense of Marr, as reviewed and questioned in [33]. At the difference of Marr's original three implementations, algorithmic and computational levels, modern vision, which is presented for instance in [30] in a comprehensive review, considers (i) the implementation level, more precisely a biophysical representation in the wide sense, and considers (iii) as the highest level the cognitive behavior, while (ii) the computational middle layer includes algorithmic aspects. This also allows us to clarify that the numerical aspects mainly stand on the biophysical side, while symbolic representation mainly stands on 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.

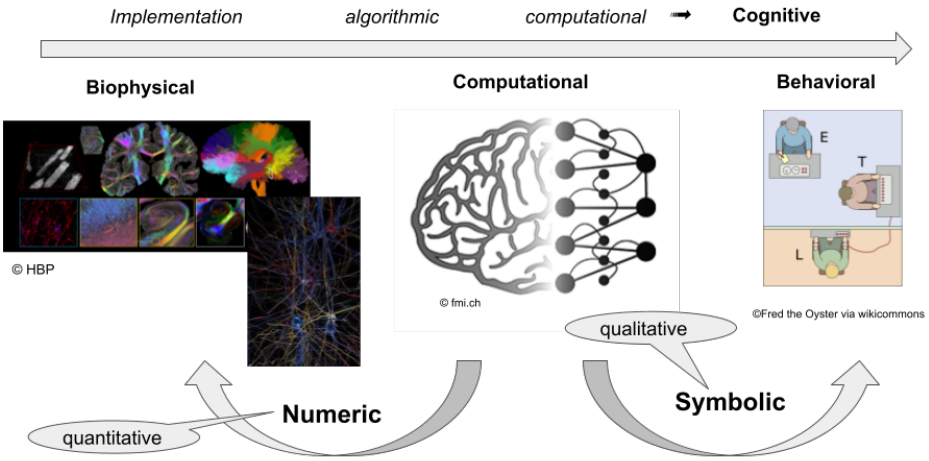


Fig. 2. A revised view of the Marr paradigm, considering modern development in computational neuroscience.

Usually, the symbolic aspects of brain activity modeling are discussed at a phenomenological level, in a human-readable form. The formalization is often represented through block diagrams, showing the relations between model

entities, each entity being defined via a key-word, a human-readable comment, properties and relations with other entities. This is more or less 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 [29]. The brain anatomy has already been formalized for a rather long time, as reviewed in [22], while macroscopic cognitive models correspond to work in progress: problem-solving tasks have been addressed by [25], and focusing recently in computational thinking, [38] made a preliminary work on the subject, considering a subject engaged in a creative open problem-solving activity as reported in [42].

This includes obviously data measures both quantitative and qualitative, as further illustrated in the sequel.

Please refer to [15] for a general introduction on ontology and to [46] for a practical approach, at the modeling level.

Modeling explicitly the brain using symbolic representations, contrasts with the use of very general deep-learning "black-box" tools, which is really questionable, as argued by [55], raising a "free-lunch´´ principle at the modeling level: what predicts everything, predicts anything.

1.4. Context: Symbolic data representation using signs

Let us now discuss what we attempt to formalize at a more theoretical level.

What is a symbol? Apparently obvious, this notion is the ground of our human knowledge, and when considering "symbolic representation", we need to clarify in detail what is defined here. 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 wide sense. It has a "meaning" when it is semantically *grounded*, in the sense of [34], who introduces this notion, as reviewed and discussed in [59], who put this notion in the context of more recent cognitive studies. This is the *"symbol grounding problem"*[3], understood as the process of embedding symbolic computations onto sensorimotor features of real live objects and behavior. This provides a semantic interpretation of the symbolic system, or model (in the sense of a model, defined as a set of logical assertions), which involves the capacity to pick referents of concepts and yielding to the notion of consciousness, as discussed in [6]. As targeted by these three references, this includes *affordance*[4], which are not only real object features but also the capability of interaction with it, in order to attain an objective, and receive some outcome.

Then at the cognitive process level, the dual problem is the emergence of symbols from sensorimotor indexes, as pointed out by [50], who details such process, coined as "ungrounding". Interesting enough, this is in direct link with more abstract semiotic approach as reviewed in [18], which target the emergence of how a symbolic representation emerges from a biological or any physical system in interaction with its environment. The process is enlighten by the semiotic approach, considering a notion of "sign" organized in a hierarchy of

- some "icon" built only from sensorimotor features, with features such as color or smell, providing element of likeness with the described object,
- icons at the "index" level, built by concrete relationships between given objects, for instance a weathercock indexing the wind direction and strength,
- giving rise to a "symbol" in the semiotic sense, which corresponds to abstract general relationships between concrete concepts or sensorimotor features, this with a qualitative break-up regarding concrete object features, for instance, a road sign, which features no more correspond to the concrete action to undergo.

A step further, at the computational neuroscience level, [2] describes the systemic functional aspect of the brain processes as interactions between three kind of memories: (i) sensorimotor distributed, in a nutshell, in the parietal and medial cortex, (ii) episodic memory in relation with the hippocampus, and (ii) semantic memory in the pre-frontal and frontal cortex. The "index" layer is related to hippocampus structures for several reasons: one one hand, the hippocampus does not store object features, but indexes toward sensorimotor areas representing the given features. It also stores temporal, spatial and relations between features, while it feeds the semantic memory. The

[3] https://en.wikipedia.org/wiki/Symbol_grounding_problem
[4] https://en.wikipedia.org/wiki/Affordance

brain semantic memory is also described by the author, like in semiotic, as abstract general relationships, including action rules. All this is a very active research field nowadays.

Such an organization, as coined by the previous author, corresponds also to algorithmic cognitive architectures, such as the [47] architecture for creative open-ended problem solving. Her architecture includes a feature level, as other framework presented previously, while ungrounding is also organized in two layers, one with relation between allowing the emergence of concepts, the upper layer being more "structural", with problem solving rules, as attributed to the brain semantic memory. However, beyond the need of two layers hierarchy above the sensorimotor layer, and very similar representation properties, the mapping is not obviously one to one, but varies with the modeling aspects or the architecture application target.

The important point here is the need of such symbolic representations, in very complementary research domains, and it was important to come to this level of details, to account for our development. These multi-disciplinary correspondences are schematized in Figure 3.

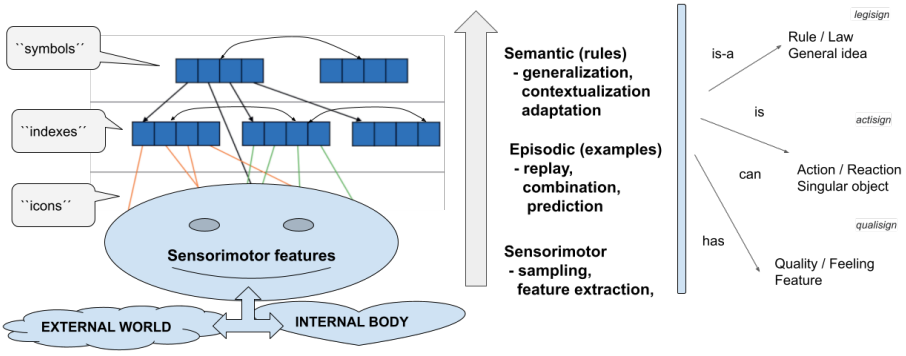


Fig. 3. The semiology hierarchy of signs and symbols, in relation with cognitive neuroscience memories.

A more concrete description of such specification is given in the next section.

2. Symbolic data specification

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

Given the previous considerations about the context and the motivation of such a symbolic approach, let us now present which kind of representations is targeted here, following [39].

The aim is to manipulate the internal symbolic representation of knowledge of the form shown in Figure 4, as introduced in [**?**]. Concepts are represented as a hierarchical data structure, in the sense of [23], how made the distinction between associative, sequential and hierarchical cognitive memories. In our context, associative and sequential memorization structures are particular cases of hierarchical data structure: they correspond respectively to "map" (also called "dictionary") and "list" at the programming data structure level.

As discussed previously, in Figure 4 examples, concepts are anchored in an input/output, that is stimulus/response. They form a sensorimotor feature spaces (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 completed by meta-information, not explicitly manipulated by the agent, but used for process specification or interpretation (e.g., here, the weight unit and bounds).

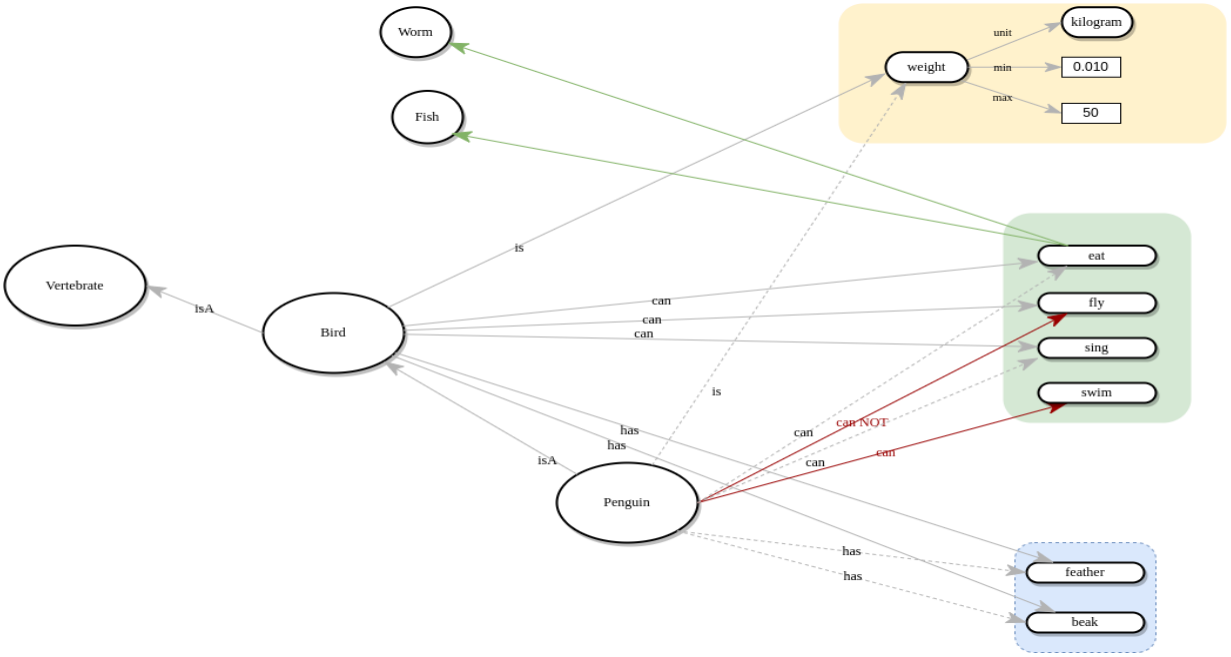


Fig. 4. Hierarchical data structure representing concepts. From [**?**].

This corresponds to the [31] approach, how also proposed the simple idea that an individual can be defined by "feature dimensions" (coined "quality" in the paper), corresponding to attributes with some typed value. For instance, a bird knowledge could correspond to what is represented in Fig 5.

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 [37]). This is a typical framework, sufficiently general to represent large fields of knowledge, in the field of cognition.

The illustrative example in Fig 5 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*[5]. 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`).

As already reviewed, using Vector Symbolic Architecture implemented at the neural spiking assembly level thanks to the Neural Engineering Framework [24], such a cognitive symbolic data structure can be implemented as biologically plausible memory [**?**].

Such a data structure defines a "concept" in the sense of [31] (e.g., here, "a bird") quoted before. It is a convex region of the state space (e.g., here, the region of all birds) in compliance compliant with the author formalization proposal. 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 [23]. At the programming level, it is going to be implemented as a "type". At the geometric level, data value corresponds

[5]https://line.gitlabpages.inria.fr/aide-group/wjson

```
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. 5. An example object defined via its features.

to points and concepts to "regions", but with a tricky property: the data structure describes also the prototype of a region.

When defining such data structure, there are several design choices. On one hand, it is always better to decompose the information as much as possible in *atomic irreducible elements* (e.g., here, `family_name: Smith first_names: [John Adam]` instead of `name: 'Smith, John Adam'`) for algorithmic processing. On the other hand, it is always better to organize features in sub-structures than to present flattened information (e.g., here, create a sub-structure for the name, birth date, etc...) to maximize modularity. We already mentioned the importance of providing as much as a possible default value, and this is a design requirement at several levels, see for instance appendix B for a discussion at the numeric data representation level. When appropriate, defining bounds is also a precious meta-element. We also point out, at the very concrete implementation level, that it is always preferable to choose explicit and standard names for features, considering already established vocabulary, thus avoid acronym or abbreviation, but choose the most common word for the feature to name.

2.2. Structuring the state space with data type

Given the previous computational objective, the basic ingredient is to generate a metrizable symbolic data structure embedding (say, a "symboling"). In the resulting metric space, the lever is the notion of editing distance. Thanks to it, a symbolic data value is step by step edited to equal another value, as detailed in the next subsection. Each elementary editing operation has an additive cost, yielding a well-defined distance and some 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 B.

Moreover, given a data type, this specification includes the requirement of defining a projection of a data value in the neighborhood of the data type region onto it.

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.

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" for the JSON syntax, "dictionary" for the Python language, or "map" with string key in another programming languages, and even "associative map" in other contexts, thus being really a universal construct.

Specification details and type parameterization are detailed in appendix A, with a description of the related methods using standard algorithms.

string[6]	This data type specifies a subset of strings.
modal[7]	This *data type*[8] specifies a level of truth between -1 (false), 0 (unknown), and 1 (true), see appendix C.
numeric[9]	This data type specifies a numeric value with its related metadata (bounds, precision, unit, ...), see appendix B.

Table 1
Basic data types.

enumeration[10]	This data type specifies an enumeration of other values.
*list-of-**[11]	This data type specifies an ordered list of values.
*set-of-**[12]	This data type specifies an unordered set of values.
record[13]	This data type specifies a record of values.
value[14]	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*[15], used for instance in *OWL*[16]. This includes structured data types such as (i) date, time and duration, (ii) *IRI*[17] including *URI*[18], thus *URL*[19], (iii) *geolocation*[20], (iv) *human language tags and locale*[21], 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 defined 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*[22] implementation provides such a parsing mechanism to and from string representations.
- New types can be defined *combining these ingredients, given specific parameters*[23] or *deriving new types*[24].

Based on this design choice we consider that a data structure is the iterative combination of such scalar and compounded data types, and are going to define editing distance and projection on such data type.

2.3. Defining an editing distance

The editing distance is defined considering *editing operations* each with a positive cost: insertion, deletion or modification of an element. Each cost can be parameterized, so that such user-defined costs allowing to model the data space taking a priory application knowledge into account: 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.

[15] https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405
[16] https://www.w3.org/TR/owl2-primer/#Datatypes
[17] https://en.wikipedia.org/wiki/Internationalized_Resource_Identifier
[18] https://en.wikipedia.org/wiki/Uniform_Resource_Identifier
[19] https://en.wikipedia.org/wiki/URL
[20] https://en.wikipedia.org/wiki/W3C_Geolocation_API
[21] https://www.w3.org/TR/ltli
[22] https://line.gitlabpages.inria.fr/aide-group/symbolingtype/RecordType.html#getValue
[23] https://line.gitlabpages.inria.fr/aide-group/symbolingtype/Type.html#declaration
[24] https://line.gitlabpages.inria.fr/aide-group/symbolingtype/Type.html#derivation

• It is obviously symmetric, because each editing operation has an inverse: deletion versus insertion, or reverse change.
• The triangular equality is also obvious because if you edit A to obtain B and then edit B to obtain C, you must not use less operation than to relate directly A 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 allows the definition of a distance between two inputs, but produce also a list of intermediate data structures between the two. This is, up to our best knowledge, something new regarding editing distances.

An important point is that an editing sequence is itself easily represented as a data structure, namely an ordered list of actions.

At the data representation level a complete data structure is a "forest": A set of *disjoint tree*[25]. Furthermore, the features of a given data structure are semi-ordered. This means that some are comparable, but not necessarily all of them. This is a very important property at the algorithmic complexity level, in order the editing distance between two data structure to have a polynomial and not an exponential complexity, as developed and shown in [48]. We also constraint that editing preserves the data structure type (for instance a list can not become a set). This means that we preserve the tree filiation, which makes it computable in polynomial time, as provided in [48], and we propose a variant of this approach here. If two data are of different non-intersecting types, complete deletion, and 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 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 6 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 in order 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 [44].

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 deletion of item_3 is a second solution, the solution of lower cost being 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.

In order to illustrate such editing distance in a concrete effective case, let us consider in Figure 7 a creative open problem solving experiment observation, using a pedagogic robotic activity, called "CreaCube" and presented in [52]. A temporal sequence of such states defines the activity representation from the subject initial situation the final success or give-up state, as a trajectory. Clearly a suitable editing distance between each state of resolution allows to represent the subject progression. More details are given in appendix .

2.4. Local structuring of the metric space

We are in a metric space, thanks to the editing distance. This space is also equipped with geodesic between two data structures, thanks to the nature of the editing distance. This path between two data structures $\mathbf{s}_1$ and $\mathbf{s}_2$ is a discrete sequence built of all intermediate data structures corresponding to a truncated sequence of editing operations of minimal cost to transform a data structure to another. The key point is that for any data structure $\mathbf{s}_k$ on

[25] https://en.wikipedia.org/wiki/Tree_(data_structure)

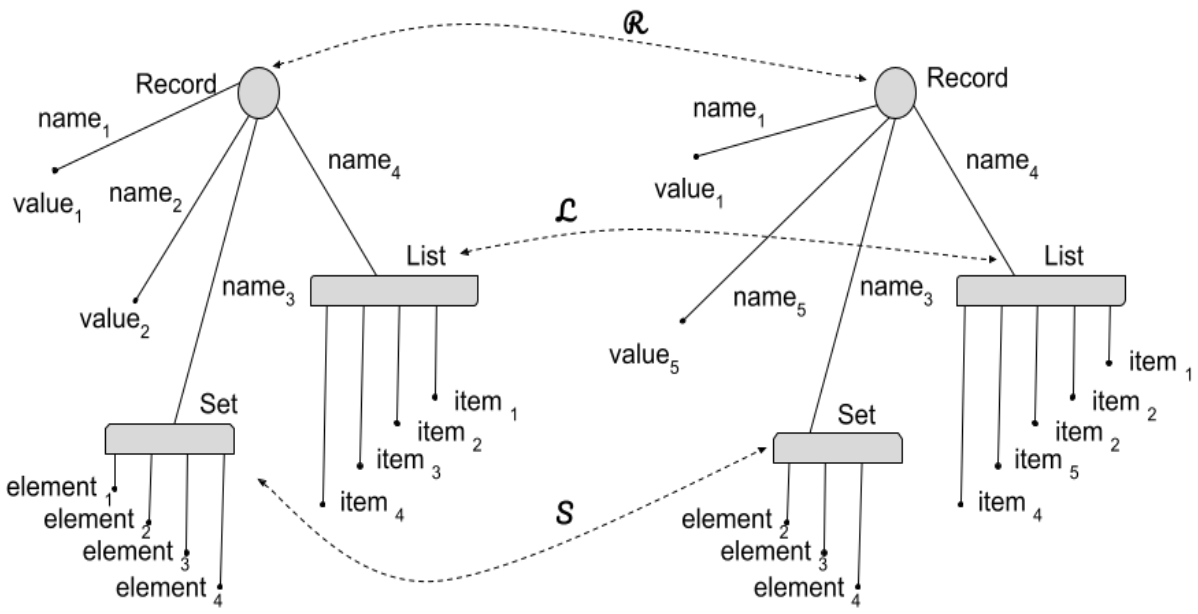


Fig. 6. The notion of edit distance, on an example including the main structured data types as illustration.

this path, by construction of the editing distance[26]:

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

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 unordered list, we first consider the association of maximal cost before the association of lower costs. Furthermore, we also need to introduce a syntactic order between any values, in case of equal cost, as detailed below.

• - For records, we consider the transformation in the item order, starting by modifying the first. This is coherent with the fact that at the semantic level, the item order is taken into account.

Such choices seem reasonable but are somehow arbitrary. At the semantic level, such choices may have an impact,

[26]**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.

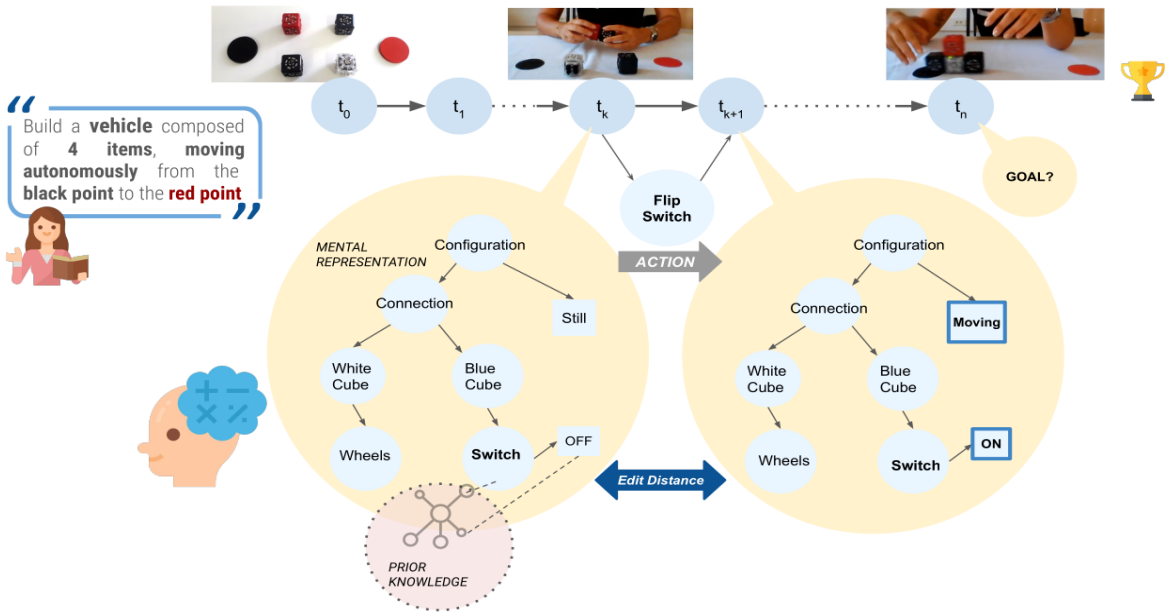


Fig. 7. An example of editing distance in a creative open problem solving experiment observation, from [39].

but it is easy to adapt them, in order to correspond to the expected 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.

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 easily verify that our design choice of a minimal geodesic is stable, in the sense that the geodesic segment corresponds in the three cases (list, set, and record) to a geodesic that complies with the corresponding uniqueness requirement.

A step further, such path is computed in order to obtain minimal distance between two path elements. This is very interesting because by weighting the distance in order to correspond to some application dependent requirements, this will allow to adjust the chosen geodesic among all possible choices.

It is important to note that the type of a list or of a set must include the empty value when deleting or inserting and this empty undefined value is nothing but 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 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 important consequences in the sequel. Beyond, though not useful for our developments, the theoretical study of such a metric structure would be of some further interest.

2.5. *Using data type to specify concepts*

In order to structure the data, the basic construction is the notion of *Type*[27], as *defined in computer science*[28]. The set of values of a given type defines a metric subspace, a region of the state space at the geometric level, equipped

[27] https://line.gitlabpages.inria.fr/aide-group/symboling/Type.html
[28] https://en.wikipedia.org/wiki/Data_type

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, list and set, must be list or set of given type. This is not an obstacle: If, say, a list must contains element of a finite set of types, then a super type including this types, is to be defined. At the geometric level, this means a region including all types regions. Doing this, two points of different disconnected types are always at 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 an union of region is convex. This is an extreme case and if two types have an intersection, the geodesic between two points of different intersecting types does not have to include the empty value.

A step ahead, all data values belong to a super a trivial super-type "value", without any further information, so that two values are at "zero" distance, thus with empty geodesic, and they project on the "empty" value with no information.

In practice, a derived data type is built from compound data types, with the specification of type's parameters, for instance numerical bound 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, in the sense defined before and introduced by [31], and the implementation design choices, detailed in appendix A. As expected by [31], a region is convex in the sense that along a geodesic between two values of a path, all values are of the same type[29].

It is also rather straightforward to verify that a region forms a compact, complete, and path-connected metric space, thanks to convexity.

Furthermore, given two data structures the editing distance is bounded since to transform one to another, we always can delete all features of the former and insert 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. Introducing reasonable assumptions on the editing distance, this semantic connectedness defines a partition of the data structure space. Generally we consider that modifying a value is always less costly than deleting and inserting it, because deletion or insertion corresponds to modifying the value to or from the empty value, which requires likely more operation, than modifying a subset of the date elements.

The proposed representation can also borrow from the usual data structure representation the notion of "schema". A schema defines a set of data structures that verifies some constraints, for instance, whether a feature has to be defined or not, or for which feature is of a given data type. This corresponds to well-established *XML-Schema*[30], or *JSON-Schema*[31]. 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 true if and only compliant with the schema, but 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 to define a projector to map a non-compliant data structure onto the closest compliant data structure with respect to the editing distance, providing also the corresponding editing sequence. More precisely, in 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 bound, the projection on the closest

[29] **Type region convexity:**

- 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 guarantied.
- 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 type. For instance, intermediate values between two positive numbers are positive values, intermediate values between two strings constrained to verify a regular expression, must be strings also verifying this regular expression, and it is also the case for derived compound types.

We thus see that type region convexity is a verified property for basic scalar and compound types, but a design requirement for derived types.

[30] https://en.wikipedia.org/wiki/XML_Schema_(W3C)

[31] https://en.wikipedia.org/wiki/JSON#Metadata_and_schema

bound provides a straightforward solution. List and set can either be empty by default, or equal to a list or set with default value.

Such a projection is a fundamental tool to manipulate these symbolic data structures at a geometric level. It is not obvious that such a projector can be easily implemented at the algorithm level, and it has to be specified for each data type. However, it appears that we are able to properly define it for all basic data type under consideration here, thanks to the notion of geodesic introduced previously, and as enumerated in more details in appendix A.

2.6. Semantic and syntactic type regions

Such a projector is not necessarily defined everywhere in the data space: There very likely values are which have nothing to do with the data type, so that "projecting" simply means deleting it and replacing by the region *prototype* which is a data defined only from default value. We thus consider two super regions of a given region:

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

The syntactic super region of any type is the whole state space, projecting onto the type prototype.

The key point is that to be able to compute a projection in a neighborhood of the semantic type region onto it, the value must fulfill some syntactic constraints for the calculation to be properly 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 bound, the projection on the closest bound provides a straightforward solution. As a counterexample, if a string can not parse to 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, value syntactically valid being semantically valid concerning this super-type. Thanks to this, the distance from a value to the type region can be calculated concerning the super-type metric, providing that the projection corresponds to the shortest distance.

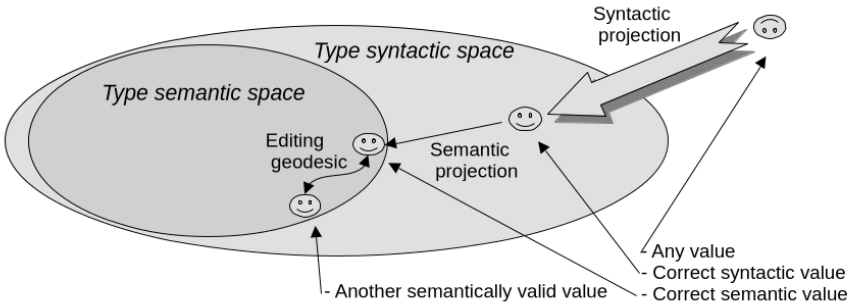


Fig. 8. Illustrating the notion of syntactic and semantic type.

We also have noticed 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 just a syntactic requirement or corresponds to a semantic meaning. Strings are alphabetically ordered, numeric (including modal) values are obviously ordered, list are ordered by the element order, 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 set 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.

2.7. Computing extrapolation away from a neighborhood value

Another issue is to extrapolate the path from a value concerning another one. 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 finding "creative" solutions (in the wide sense).

The caveat is that we do not have a priory such a mechanism. Let us consider, as an example, the standard *Levenshtein editing distance*[32]. It has no well-defined "inverse" If we consider the distance between `mama` and `mamie` we easily obtained `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 very little chance that when inserting, deleting, or changing any letter we draw a syntactically correct word, and even less chance to generate a semantically useful word. More generally, for more complex symbolic data structure, simply generating unconstrained random values around a current value, by some simple random draw, there is almost no chance to generate a useful or even meaningful random value.

Using the data type bounds

We thus have to design this new functionality, given the existing ingredients. We will get around with the idea of choosing some alternative "escape" path towards a data structure away from the target. Since, by construction, the data space is bounded as soon as sufficiently specified, as illustrated in Table 3, we will rely on such bounds.

string[33]	Bounded if defined by either an enumeration or by a bounded *regular expression*[34] or a *maxLenth* meta-value bound.
modal[35]	Bounded in $[-1, 1]$.
numeric[36]	Bounded as soon as the *min* and *max* meta-value are defined.
enumeration[37]	Bounded by construction.
list-of-$*$[38]	Bounded as soon as a *maxLength* meta-value is defined.
set-of-$*$[39]	Bounded as soon as a *maxLength* meta-value is defined.
record[40]	Bounded as soon as the record items to take into count are explicitly enumerated.
value[41]	Bounded by the fact that it is only the undefined value as explicit value.

Table 3
Basic and compound data 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.

Of course, 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.

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

[32] https://en.wikipedia.org/wiki/Levenshtein_distance

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 9. Choosing the best extrapolation path is an under-determined problem. Let us discuss how to better specify it.

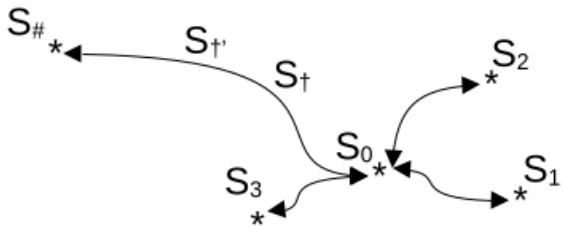


Fig. 9. Describing the extrapolation issue of a data $\mathbf{s}_i, i > 0$ through $\mathbf{s}_0$, given a bound $\mathtt{S}_\#$, suitable for $\mathtt{S}_1$ extrapolation, while less adapted to $\mathtt{S}_2$ and $\mathtt{S}_3$ extrapolation.

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.

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 linear combination of distances.

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 [42] $\nu(\mathbf{s}_\dagger)$ increases with the distance $d(\mathbf{s}_0, \mathbf{s}_\dagger)$ to $\mathbf{s}_0$, while obviously $\lim_{\mathbf{s}_\dagger \to \mathbf{s}_0} \nu(\mathbf{s}_\dagger) = 0$.

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 simply 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 which is a bound on the acceptable geodesic triangular inequality value.

Actual available implementation

[42] **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:

$$\nu_{i0}(\mathbf{s}_{\dagger'}) - \nu_{i0}(\mathbf{s}_\dagger) = d(\mathbf{s}_i, \mathbf{s}_0) + \underbrace{d(\mathbf{s}_0, \mathbf{s}_\dagger) + d(\mathbf{s}_\dagger, \mathbf{s}_{\dagger'})}_{d(\mathbf{s}_0, \mathbf{s}'_\dagger)} - d(\mathbf{s}_i, \mathbf{s}_{\dagger'}) - d(\mathbf{s}_i, \mathbf{s}_0) - d(\mathbf{s}_0, \mathbf{s}_\dagger) + d(\mathbf{s}_i, \mathbf{s}_\dagger)$$
$$= d(\mathbf{s}_i, \mathbf{s}_\dagger) + d(\mathbf{s}_\dagger, \mathbf{s}_{\dagger'}) - d(\mathbf{s}_i, \mathbf{s}_{\dagger'}) \geqslant 0 \text{ by the triangular inequality.}$$

The actual *available implementation*[43] 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 of a given 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 former method being more efficient if we select a small subset.

Complementary specification of the extrapolation

A step further, to attempt to better specify 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.

• One on the distance, selecting only target $\mathbf{s}_\dagger$ with[44]:

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

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

• One on a generalization of the orientation, selecting only target $\mathbf{s}_\dagger$ with[45]:

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

e.g., $\mathbf{s}_1$ or $\mathbf{s}_2$ but not $\mathbf{s}_3$ in Figure 9, thus which a locally an obtuse angle and not an acute angle.

These distance and orientation complementary constraints are useful to verify the coherence of the proposed solution or to reduce the amount of search along the geodesics.

3. Defining Scalar fields of symbolic data structures

3.1. Position of the problem

When considering usual algorithms such as variational approaches used in supervised learning or clustering used in unsupervised learning, we need to associate a numerical value to data structures, usually called a cost value or a 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.

---

[43] https://gitlab.inria.fr/line/aide-group/symbolingtype/-/tree/master/src

[44] **Extrapolation in the Euclidean case:** In the Euclidean case, for any point $\mathbf{v}_k \stackrel{\text{def}}{=} \alpha\,\mathbf{v}_0 + (1-\alpha)\,\mathbf{v}_i, \alpha \in [0,1]$ 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, through the metric space does not enjoy an explicit property of "1st order Euclidean space", as for manifolds.

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

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

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 locally reused in our case.

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:

$$f : \mathcal{S} \hookrightarrow \mathcal{R}$$
$$\mathbf{s} \to r$$

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.

3.2. 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 we only have a metric space equipped with a distance, we define such a neighborhood which 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 $d_{\max}$.

which is a very standard way of defining neighborhoods, for instance when considering operations on manifolds, as for instance developed in [13], or in the context of dimensional reduction or graph manipulation, as used by [49] in the context of graph clustering, quoting here two very different examples to illustrate the common use of such neighborhood definition.

Given such a neighborhood, three cases are considered:

- If $K = 0$ the value is undefined.
- If $K = 1$ we only can 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 10.

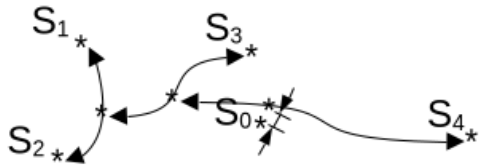


Fig. 10. As graphically represented here, $\mathbf{s}_1$ and $\mathbf{s}_2$ are replaced by the value on their geodesic as closed as possible to $\mathbf{s}_0$, which is then related to $\mathbf{s}_3$ by a geodesic, making the same choice again, which finally is related to $\mathbf{s}_4$, yielding to 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 the intermediate data structure which is as closed as possible to the data structure value to approximate is a reasonable choice, in this context, and yields a linear algorithm in terms of complexity.

At this stage, our method is sub-optimal since we only consider one geodesic and not all possibles 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 but an interpolation with respect to bounds.

We can not explicitly consider that the data structures live in an abstract manifold, which is a space that is locally Euclidean up to the 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 to each other, and only consider geodesic which are entirely included in the considered region. The latter method relies on an arbitrary kernel $\nu(d)$, take into account points that could be less closed to the point to interpolate, thus introducing a higher bias, and does not guaranty that the point is included in the considered region.

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

3.3. 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[46]:

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

which corresponds to a generalization of a barycenter, as it is required for instance in clustering algorithms, such as *K-mean clustering*[48].

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 If $K = 2$, it is a reasonable choice to minimize, for a given exponent[49] $e > 0$, $l = w_1\, d(\mathbf{s}_0, \mathbf{s}_1)^e + w_2\, d(\mathbf{s}_0, \mathbf{s}_2)^e$ on the geodesic between $\mathbf{s}_1$ and $\mathbf{s}_2$, because $d(\mathbf{s}_0, \mathbf{s}_1) + d(\mathbf{s}_0, \mathbf{s}_2) = d(\mathbf{s}_1, \mathbf{s}_2)$ is minimal[50]. This is easily performed by scanning the different data structures to find the minimal value.
- 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

---

[46] **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*[47] of the case with data structures, where we only can consider distances.

[48] https://en.wikipedia.org/wiki/K-means_clustering

[49] **Minimum as a function of the exponent:** We left to the reader to verify that, if $e \leqslant 1$ the minimum is on one geodesic extremity, while if $e > 1$ there is a unique minimum between both extremities, while for $e = 2$, we obtain explicitly $w_2\, d(\mathbf{s}_0, \mathbf{s}_1) = w_1\, d(\mathbf{s}_0, \mathbf{s}_2)$.

[50] **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:

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

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.

two former data structures by the latter, thus decreasing the set of data structure cardinal by 1, this being iterated until $K = 2$.

This again only a sub-optimal strategy, since we only consider a linear number of geodesics and not all possible ones, but this provides an approximate estimation, using only local operations.

4. Illustrations

At this preliminary stage of development we are not able to report large scale experimentation, but would like to share some interesting toy experiments that can already allows us to evaluate and illustrate the proposed mechanisms.

4.1. 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*[51] in image processing. This is quite interesting here, because it allows to visualize a geodesic path for some examples, as reported in [7].

Our collaborators[52] have designed two morphing demos.

Visual morphing demo An *https://line.gitlabpages.inria.fr/aide-group/symbolingtype/visualmorphing/titi-toto.html*[53] image morphing demo, using the *SVG symbolic description of a drawing*[54], allows one to reproduce pixelic image morphing, for some vectorial drawing, as shown in Figure 11. The interest point, at the validation level, is that the present formalism can be applied also to realistic open standard, such as SVG. What the geodesic path explicitizes is one (over likely several) optimal sequence of drawing editing operation to turn from the former to the latter drawing. The subset of the SVG standard in use in this demo is *defined as a symboling type*[55]. Given this, vectorial drawings are manipulated as any other data structure. Please refer to the demo link[??] for a detailed observation of the produced image sequence. This preliminary result shows how the symbolic data structure defining the drawing elements is changed, mainly from top to bottom, because it is the record item order, which is an arbitrary choice induced by the geodesic selection among all possible geodesic. As mentioned previously, the fact the editing distance is weighted by any user suitable parameters, and the fact that the geodesic path is defined by local change of minimal editing cost, will easily allows a user to tune such 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, each element operation being counted for 1, which seems really high with respect to the editing operations, such as change or move 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 toy but realistic data set, the number of elementary operations becomes easily high.

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*[57], also *translated as a symboling type*[58] for the standard subset used here, they have reproduced the same calculation, as shown in Figure 12, while a complete demo result is available[??]. Since the former song uses tied notes, while the latter uses dotted notes, it easy to see which part of the scores turns from *https://line.gitlabpages.inria.fr/aide-group/symbolingtype/musicmorphing/au-clair-de-la-lune-frere-jacques.html*[56]

[51] https://en.wikipedia.org/wiki/Morphing

[52] This illustration has been developed by Paul Bernard, Benjamin Hate, and Morgane Laval during their engineering curriculum, under the supervision of Chloé Mercier.

[53] tititoto

[54] https://en.wikipedia.org/wiki/SVG

[55] https://line.gitlabpages.inria.fr/aide-group/symboling

[56] auclairdejacques

[57] https://en.wikipedia.org/wiki/MIDI

[58] https://line.gitlabpages.inria.fr/aide-group/symboling

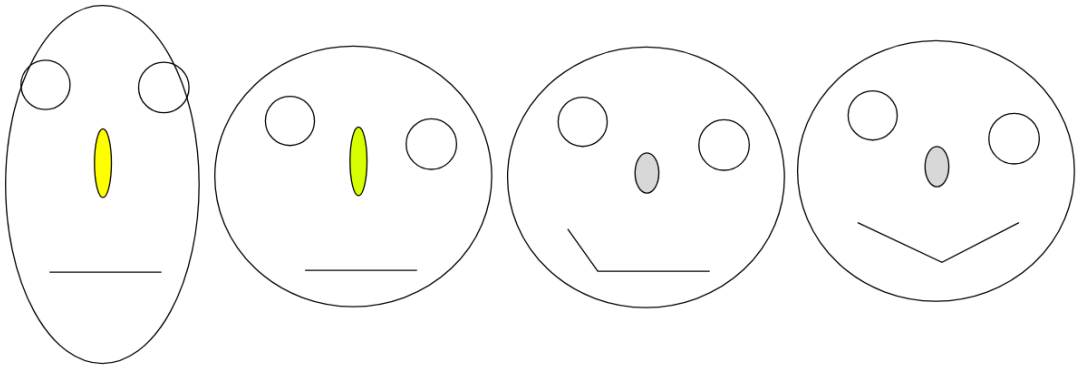


Fig. 11. A subset of the morphing between the “titi” drawing on the left and the “toto” drawing on the right: two intermediate images on the geodesic path defined by the SVG drawing representation are shown.

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, each element operation being counted for 1, thus an average of about 450 operations between two consecutive element on the path. Again this may seem dizzying, but this is at the scale of this not so trivial data set.

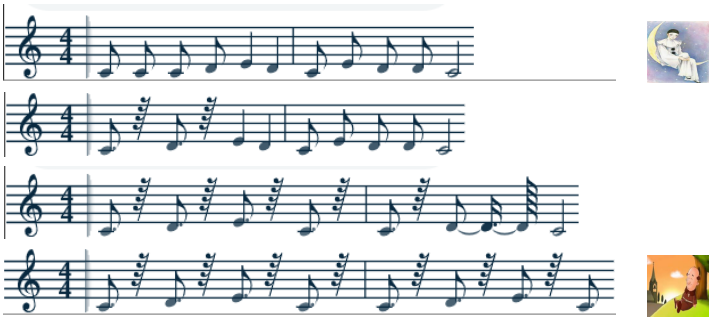


Fig. 12. A subset of the morphing between the “au clair de la lune” and the “frère jacques” early parts of the songs: two intermediate scores on the geodesic path defined by the MIDI music representation are shown.

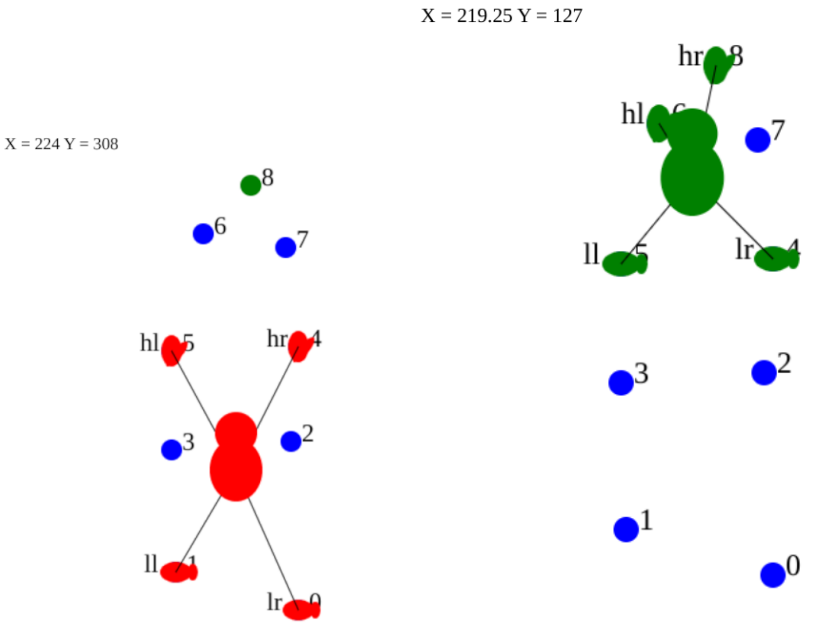
4.2. A symbolic problem solving setup

Beyond simply observing the path between two data structure, we have also considered *stating a problem solving setup*[59][60]. 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 here to evaluate to what extents it could be easy to model both the climbing wall and the climber, including with constraints on limb length and posture. The result is shown in Figure 13. The *complete experimentation available on line*[61] allows to connect the toy setup to an external algorithm of trajectory resolution, here using a `Python` standard genetic algorithm. This, by the way, also shows that the middleware developed in `C/C++` is easily usable from `JavaScript` at the web interface level and from `Python` at the algorithmic and data analysis levels. Still very preliminary, this illustrates how we can target problem-solving issues at the symbolic level.

[59] https://github.com/Tpris/Projet_semestriel_symbolic_RL

[60] This illustration has been developed by Thibaut Lanier, Léo-Paul Mazière, and Priscilla Tissot during their engineering curriculum, under the supervision of Axel Palaude.

[61] https://tpris.github.io/Projet_semestriel_symbolic_RL




```
checkState Error:
        ! hright below lright
 state:
{
  "limbs": {
    "hleft": {
      "x": 163,
      "y": 289,
      "h": 3
    },
    "hright": {
      "x": 289,
      "y": 473,
      "h": null
    },
    "lleft": {
      "x": 167,
      "y": 405,
      "h": 1
    },
    "lright": {
      "x": 283,
      "y": 441,
      "h": 0
    }
  },
  "body": {
    "x": 225.5,
    "y": 402
  }
}
```

Fig. 13. 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).

4.3. Reinforcement learning in a survival experiment

Here we consider the application to usual machine learning algorithms, for instance, reinforcement symbolic like learning mechanisms, as introduced by [41]. Reinforcement learning is a sub-field of machine learning that is concerned with how agents learn to make decisions in an environment in order to maximize some notion of reward. It has shown a 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.

The add-on of this work[62] [51] is to consider a simple but formal ontology, as represented in Figure 14, and for what concerns the symbolic description of food, in Figure 15. The agent feels internally its mood, health level, and hunger, ranging from negative (e.g., sadness) to positive (e.g., joy) values. It enjoys sight, taste, touch, and smell senses, defined by qualitative values. The external environment influences internal states, as function of the encountered entities. The agent choose an action among a discrete set (e.g., eat, attack), and the agent's parameter's values are updated accordingly. There are 256×25×9 = 57600 possible values (number of internal states × number of entities × number of actions). The ontology structures all entities, states and actions in a class taxonomy so that what has been learned can be reused for related elements in a given neighborhood.

The implementation is *available as a documented open software*[63], while 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 in order to learn the expected reward and thus generate a correct strategy,

[62] This illustration has been developed by Waris Radji, Corentin Léger, and Lucas Bardisbanian during their engineering curriculum, under the supervision of Chloé Mercier.

[63] https://github.com/riiswa/symbolic-rl/

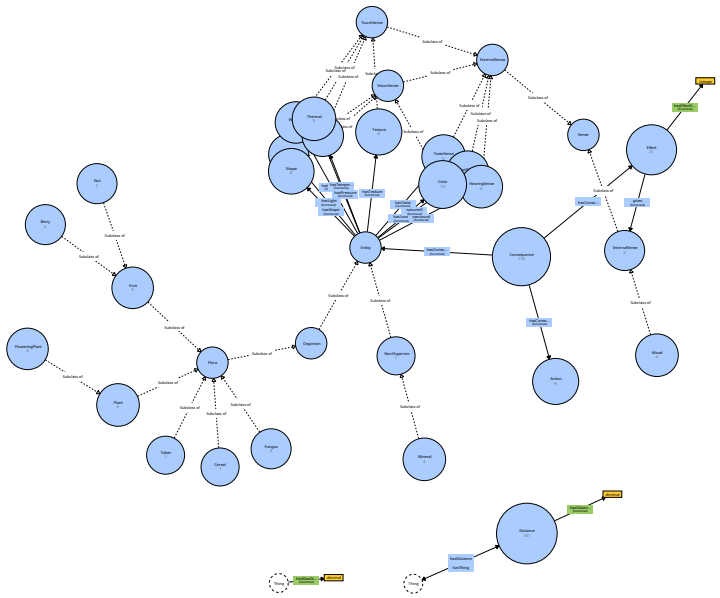


Fig. 14. The ontology used to describe a simplified external and internal environment in which a agent survives, from [51].

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 very common *Q-Learning algorithm*[64] at the numerical level using a well established *AI Gym, now Gymnasium*[65] middleware. They have implemented the [41] symbolic Q-Learning algorithm, which is based on computing the editing distance, as developed here, in order to evaluate the reward on one point as a exponentially decrease weighted sum of known rewards in the neighborhood, thus making use of the state space metric. The key meta-parameter for symbolic learning is the radius ρ of the neighborhood, weighting the exponentially decrease weight.

The results shown in Figure 16 show that the use of symbolic representation can significantly improve the generalization capabilities of reinforcement learning agents, providing 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 a better performance, this quickly (after about 2000 steps, in comparison the standard approach that has not started to converge after 10000 steps).

However, as the authors conclude, the ρ hyperparameter, which scale the editing distance, is quite sensitive, yielding drastically change in performance, and exploring ways of dynamically adjusting the radius parameter over

[64] https://en.wikipedia.org/wiki/Q-learning

[65] https://gymnasium.farama.org/Open

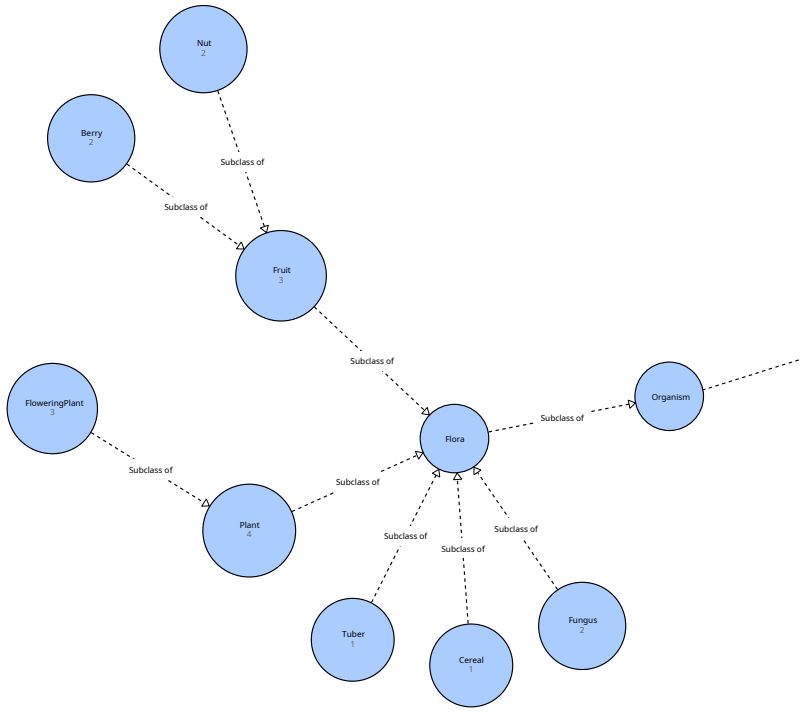


Fig. 15. A detail of the ontology regarding the symbolic description of food, from [51].

time, to learn the optimal radius value is to be considered. In fact, the issue here is more general: algorithms using the present approach, highly depend on the editing distance design.

5. Discussion

5.1. On the computer efficiency of the method

The data structure is implemented in `C/C++` using a specific data object model based on the `std::unordered_map` associative map data structure, with a `std::vector` for the record keys. We obtain performances comparable to the *N. Lohmann*[66] C++ native library written in `modern C++`, as shown in Table 4. The test has been conducted on three *Milo Yip*[67] benchmark's set: `twitter` a 13913 attributes structured JSON structure, `canada` a 167178 attributes multi-dimensional numeric array, and `cimp` a 37777 attributes data base JSON structure. The present implementation is for some operation a bit slower because the record item insertion order is maintained, for semantic reasons detailed in this paper. However, this overload is negligible with respect to editing distance computation, presented next. Both middlewares are less performing than speed optimized libraries such as *jsoncpp*[68] with minimal functionalities.

The CPU computation times are evaluated on a standard middle-cost laptop with `Intel©Core`^{TM} `i5-8265U CPU @ 1.6GHz x 8` with `16 GBiB memory` and no GPU usage.

[66] https://github.com/nlohmann/json
[67] https://github.com/miloyip/nativejson-benchmark
[68] https://github.com/open-source-parsers/jsoncpp

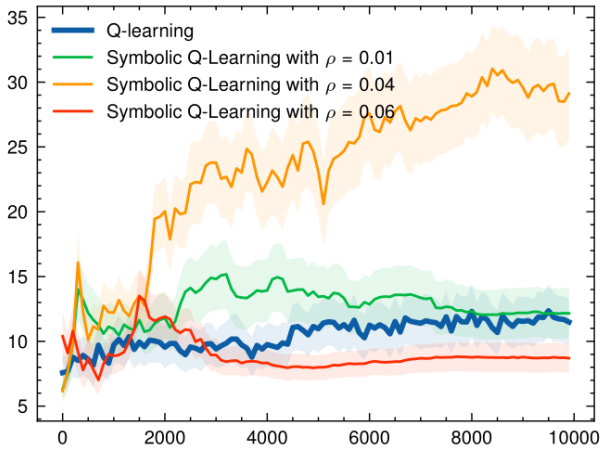


Fig. 16. Comparison between a Q-Learning and a symbolic Q-Learning cumulative rewards, considering several neighborhood sizes for the latter, from [51].

	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 computations after distance estimation represent the main computation time consumption and we obtain for the morphing demos, the results given in Table 5.

	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.

Edition distance computation scales in a supra linear way depending on the involved data structures, linearly on records, but with a quadratic and cubic increase on list and set respectively, while the path computation scales linearly with its length and the data structure attribute count. The tiny experimental observation confirms that adding the computation of the geodesic path to usual editing distance computation yield only a 5% to 10% overload.

Regarding memory consumption, using associative map and index vector, it obviously scales linearly with the attribute count, the element overhead of such associative map coupled with an index vector being of two memory slots for each attribute and a few memory slots for the related record overhead. In the present implementation the caveat is that strings are managed by copy, instead of having a string dictionary and manage them by reference. This would be an obvious improvement, and it is a close perspective of this work.

These numbers have no "absolute" meaning. They simply shows that the proposed computations have the order of magnitude of what can be computed with a modern laptop computer, without requiring large computer clusters or dedicated hardware. This also give a rule of thumb to estimate the computational cost of problems at higher scales.

5.2. Advantages and disadvantages of such metric space embedding

Some elements of comparison regarding the advantages and disadvantages of the proposed metric space against other types of groundings has been already sketched out along the paper. Let us synthesize the main elements[69].

Nowadays, symbolic data such as words of text are embedded in neuronal parametric vectorial spaces, considering statistic measures, in order to process the resulting data with black-box artificial neural networks, using specific architectures such as generative deep neural networks, such as variational transformers. This allows to obtain unprecedented capabilities in information processing performances, but at a huge financial and environmental cost, with several concerns (tech giant domination, intellectual property, ...).

The proposed approach avoids the initial embedding of symbolic data in a large dimensional vectorial space. The other way round, it directly equips the symbolic set with the required numeric tools (metric, path, projector, ...) required to apply usual a machine learning algorithms, such as clustering or trajectory generation, as illustrated previously. We thus can expect such approach to be more parsimonious, yielding results easily interpretable and explainable, in the sense of [16]. We also have pointed out that it would be more explanatory when applied to brain modeling, as discussed by [55].

Another practical aspect is ergonomic. Our experience in multidisciplinary research (here, learning science, neuroscience and computer science), such as during the *AIDE exploratory action*[70], 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 of deep practice.

And the effort to develop an open multi-platform and multi-programming language middleware should also help.

However, in practical applications, the proposed distance depends on many somehow arbitrary parameters, which are likely influencing the algorithm performances. This has been experimented in subsection 4.3 for the global distance scale ρ. More generally, the modeling choices and the number and quality of properties that are attached to each data type are crucial to derive a pertinent specification. This questions the applicability of the proposed metric space.

What has been done in subsection 4.3 may help solving this caveat, and leads to a kind of recipes:

- Following, for instance, the tutorial presentation of [46] one may start modeling and designing the problem description, enjoying the quality and richness of the ontology approach, restraining 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 extents this could be structured in a hierarchical way with record, list and set, noticing that such structuring is fully available at the ontology level.
- Considering each quantitative parameter, on should take care of all useful metadata, such as bounds and precision, as developed in appendix B: this will produce a normalization of all numerical elements when computing editing distance.
- Hopefully, by default, each data type have equal weights for each element, and a default editing distance can be computed without having to preset each parameter. Then, if some a priory knowledge allows one to modify this default choice, that could be easily done, preferentially with parsimony.

Following such a track may help designing and parameterizing the data structure.

As an alternative to ontology approach, object oriented approaches could also be considered. In any case, this approach is a complement to ontology or object oriented approaches.

[69] This paragraph, among other precious suggestions, is a direct suggestion of one of the reviewer, Lia Morra.

[70] https://team.inria.fr/mnemosyne/en/aide

Another disadvantage of the approach with respect to usual numerical embedding is the fact that the distance computation time is, for list and set elements, respectively quadratic or cubic instead of linear, as it is the case for a Euclidean distance computation after a numerical embedding. However, usual numerical embedding require high dimensional space, while in the present setup we stay at low dimensions. In relation to this, distance computation with the Levenshtein or Hungarian algorithms, are intrinsically not easy to be parallelized, and specialized processors like GPU seem to be helpless to accelerate these non trivial computations.

5.3. On the biological plausibility of the proposed knowledge representation

Although beyond the scope of this paper, it is interesting to point out that the biologically plausible of such knowledge has been considered, in other studies. Symbolic reasoning taking place in the cortical-thematic loops through the basal ganglia have been properly modeled using spiking neurons proposed in Vector Symbolic Approaches (VSA) approaches as introduced by [58], while [43] 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 Semantic Pointer Architecture scales properly to address large-scale problems, and allows one to encode data structures as defined here.

These developments include creativity tests simulation, as studied by [36] 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 [57].

Such biologically plausible numerical grounding of such data structure is entirely different from the sub-symbolic numerical grounding proposed previously, because VSA approaches stand on hyper-dimensional compact (usually a hyper-sphere) space of randomly drawn vector, while basic symbolic operations are represented by abstract algebraic operators implemented as biological network simulated distributed operators. A precise development on VSA of what has been proposed here is a research project on his own, but it is clear that the basic ingredients are here, and a recent work has precisely described how all data structures used in the present developments can be implemented in VSA [40].

Thanks to this last work, in progress, we have all the ingredients for a biological plausible implementation of what we call here “symboling” mechanisms.

5.4. Perspectives

In order 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 D. This is presented to give an overlook of the potential tools that could be summoned in extension of the present work, and also the implementation feasibility. It also includes some theoretical remarks with respect to link with other formalism.

Let us now discuss some perspectives in a few application domains.

Application to complex problem solving

We have defined problem-solving tasks at a geometric level, considering being located somewhere in a state-space, to reach some final (unique or alternative) state, finding a way from the former to the latter, while satisfying the path constraints.

This definition is computationally effective, even for complex symbolic state space in the sense that we have developed a fully formalized problem-solving algorithm, each element being precisely defined at a symbolic level. This may be used in several computational domains, such as robotic trajectory generation and optimization, transportation theory, or operation research.

We are interested in ill-posed problems which means that estimating the present initial state, choosing some final goal parameters, discovering the useful part of the state-space, and generating the trajectory have to be developed during the problem-solving task, and are somehow part of the task. This means that the conceptual representation has to be adapted during the task execution, thus completed and corrected, as preliminary experimented in [51].

Application to creative problem solving

Creative problem solving requires divergent thinking as analyzed in [53] and formalized in [4]. In a nutshell, divergent thinking requires not only interpolating but extrapolating 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 usual emu's features. Thanks to the notion of schema proposed above, here used to specify the target requirements, and the notion of a projector, we have an effective implemented mechanism to generate an extrapolated unprecedented data structure, by projecting the prototype onto the targeted 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 some randomness to explore alternatives or introduce some constraints, as in the previous example, to conform to any requirement.

Reasoning by analogy: Following the definition of analogy reasoning proposed by [32], 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 [32] which performs it considering ontology. Here we propose to implement it at our hybrid geometric level.

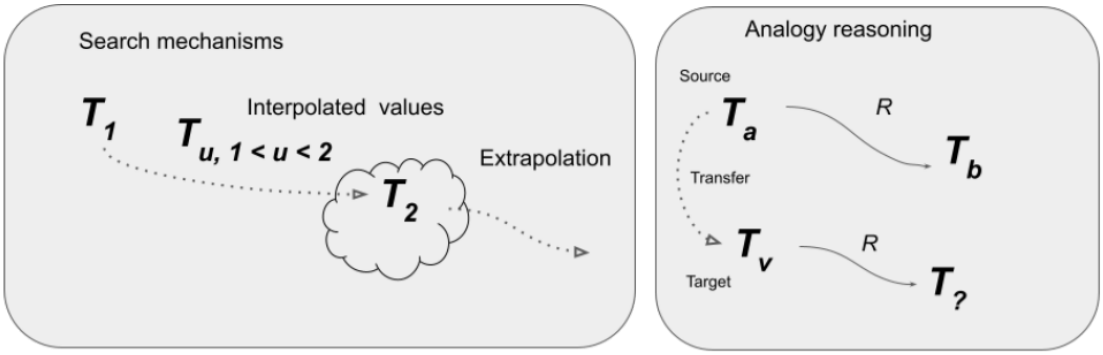


Fig. 17. A schematic representation of two kinds of generative processes. Left: search, by extrapolation in the mirror of an interpolation between two structures. Right: reasoning by analogy, as formalized in [32].

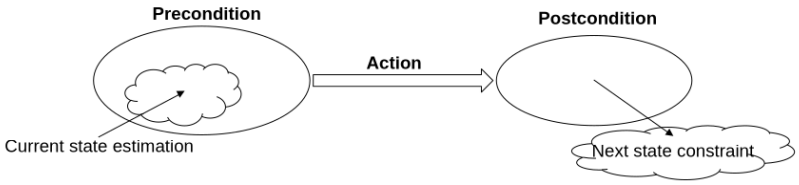
Application to higher-level resources specification

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 also can not only define positively defined knowledge but add a property to defined uncertain, approximate knowledge, with a certain belief level, regarding what is stated, as formalized in appendix C.

6. 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 very powerful algorithms, as summarized in Figure 19, showing




```
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].

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.C, regions inclusion yields a class taxonomy, while state value attributes and state value properties can be interpreted as a scalar-field on their domain $\times$ 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 [**?**].

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 define 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.

6.1. Acknowledgments and contributions

The *NAI*[71] journal reviewers, Lia Morra and Ben J. Wright, are especially and gratefully acknowledged for their deep reviewing work, yielding precious advises allowing us strongly improve this paper.

Frédéric Alexandre is deeply acknowledged for his powerful ideas at the origin of this work, his shared expertise regarding computational neuroscience aspects of this work, and his technical advice along 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 strongly contributed to the main scientific ideas of this work, co-formulate theoretical aspects, for more than 30% each, and have contributed to paper construction and writing. Thierry Viéville has rounded up these elements together and written the first draft of the paper.

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, default comparison.

[71] https://neurosymbolic-ai-journal.com

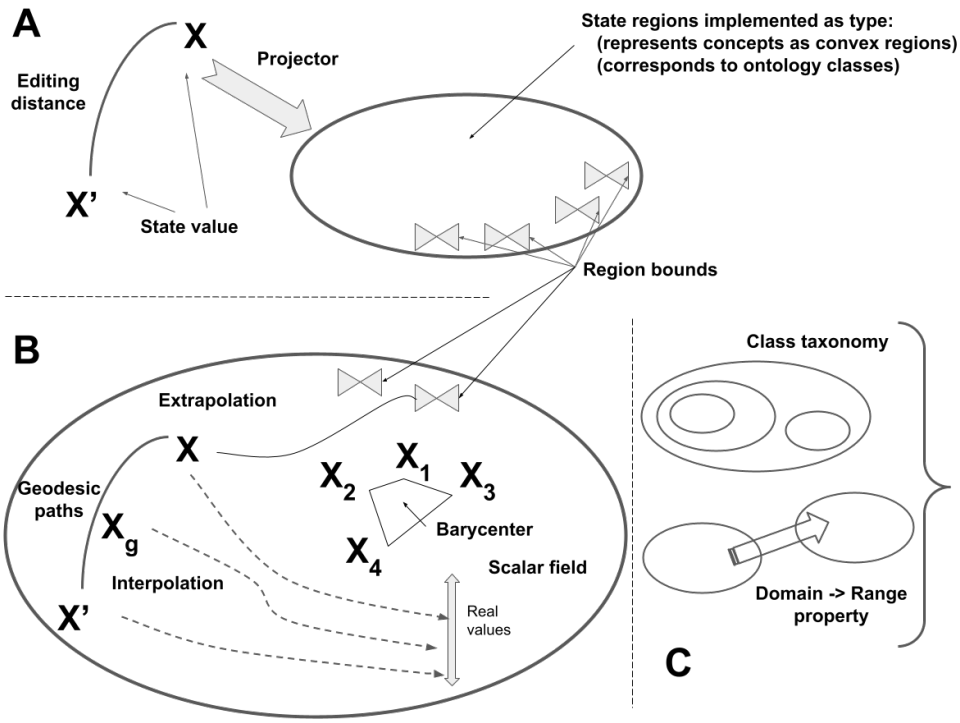


Fig. 19. Summarizing the different "symboling" tools:
A: Tools defined at the design level: the notion of a region corresponding to a type or concept, equipped with a projector, and equipped with a metric using an editing distance between the different state values, while bounds are to be specified.
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.

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 char 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.

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.

ModalType specification

– **Syntactic projection**
- Returns *NaN* if not a parsable number or a predefined string.
- Returns the zero 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 sub-type of it.

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, empty value if one record item is missing.

– **Geodesic path**
- The path is constructed in the record item order scanning simultaneously both record item lists.

– **Comparison**
  - The comparison is performed in the record item order scanning simultaneously both record item lists.
  - 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.

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.

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 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*[72] applied on the list element and delegating the edition cost to the item type distance computation [44].

  - 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 re-deriving a cost function.
– **Geodesic path**
  - The geodesic path corresponds to the *Wagner-Fischer algorithm*[73] shortest path in the matrix distance yielding to 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 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.

[72] https://en.wikipedia.org/wiki/Levenshtein_distance
[73] https://en.wikipedia.org/wiki/Wagner-Fischer_algorithm

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*[74] implementation of the matrix version of the *Hungarian algorithm*[75], 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 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. Numeric scalar data

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.

Such specification is important to properly manipulate quantitative information. In particular, the values can be normalized, which means mapped onto the $[-1, 1]$ interval, or mapped onto a finite set of relevant values. One consequence is that algorithm precision thresholds can be deduced (often using first-order approximations), spurious values can be detected, numerical conditioning of algorithms is enhanced, as proposed and applied in [65], where this specification is discussed. At the computational specification level, these parameters define a sub-type of usual numerical types, yielding a better definition of the related code.

This concerns both external numerical sensory data and internal quantitative data, having always bounds and precision. It is obvious that any quantitative measure is bounded, for instance physical velocity magnitude stands between 0, for a still object, up to the light speed. It also always given up to a given precision, for instance a localization in an image is given up to one-pixel size, or a school ruler up to 1-millimeter graduation. The key point

[74] https://github.com/mcximing/hungarian-algorithm-cpp
[75] https://en.wikipedia.org/wiki/Hungarian_algorithm

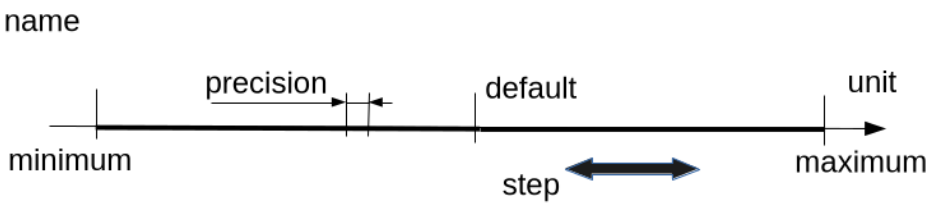


Fig. 20. Specification of a numerical value. Metadata includes a name, a unit, a default value, bounds (minimum and maximum), a precision under which two values are not distinguishable, and a step value corresponding to a neighborhood size used to cover the value range.

is that it is useful to make explicit this obvious fact at the specification level instead of using it implicitly when required, in order to be generically used in algorithms. For instance, an iterative estimation algorithm, thanks to this specification, is easily tuned up to output required precision. A step further, symbolically coded data can always be sampled. For instance, a vector font drawn on a canvas is sampled ate the pixel precision.

The notion of the positive sampling step, in order to define a local neighborhood size, is used to weight distance calculation, and to properly sample the data space: The underlying idea is that the state space is locally convex so that in a given neighborhood local search of an optimum yields to the optimal local value. This idea has been implemented in the *stepsolver*[76] variational solver, including optimizer and controller.

This specification induces a *pseudo-metric*[77]:

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

in words, the distance is weighted by the *step*, which is the neighborhood approximate size, while 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 case. This a deterministic counterpart of *statistic hypothesis test*[78], with $\mathcal{H}_1$ versus $\mathcal{H}_0$ hypothesizes.

Appendix C. Modal scalar data

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 corresponds qualitatively to the human appreciation of the degree of belief in a fact.

The true value corresponds to 1 (fully possible and fully necessary), the false value to -1 (neither possible nor necessary, thus impossible), and the unknown value to 0, which corresponds to a fully 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*[79], and is in one to one correspondence with respect to the possibility theory. Possibility theory is devoted to the modeling of incomplete information, in link with an observer’s belief regarding a potential event and surprise after the event occurrence. Please refer to [19] for a general introduction. While almost all partially known information is related to probability, human “level of truth” is more subtle and related to possibility and necessity, as formalized in the possibility theory. This is introduced and discussed in [19] and [20]. This in link with modal logic, modeling something true in a “given context” as developed by [26]. Interesting enough, this is also considered as representative to what

[76] https://line.gitlabpages.inria.fr/aide-group/stepsolver
[77] https://en.wikipedia.org/wiki/Pseudometric_space
[78] https://en.wikipedia.org/wiki/Statistical_hypothesis_test
[79] https://en.wikipedia.org/wiki/Three-valued_logic#Kleene_and_Priest_logics

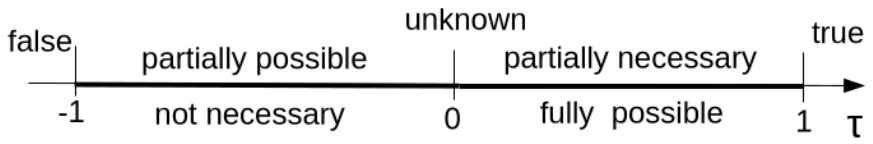


Fig. 21. Specification of a modal value of belief, this with regard to necessity and possibility, as defined in the possibility theory.

is modeled in educational science and philosophy as reviewed by [54], because it corresponds to commonsense reasoning in the Piaget's sense [56], taking exceptions into account, thus considering non-monotonic reasoning. Furthermore, in symbolic artificial intelligence, specifically considering knowledge representation through ontology, the link has been built between this necessity/possibility dual representation and ontology [60]. This must be understood as a deterministic theory, in the sense that partial knowledge is not represented by randomness. An extension taking randomness into account is proposed in [61, 64].

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.

Appendix D. Data value interfacing

In order 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. This is presented to give an overlook of the potential tools that could be summoned in extension of the present work. At this stage, we have simply check the implementation feasibility.

Interfacing with the wJSON data object model It is straightforward to notice that our scalar values map one to one to a *JSON*[80] data structure. More precisely, at the implementation level, the data object model (DOM), is a *Value*[81] which is either:

- (i) an atomic string (including string parsable as numeric or boolean value) or
- (ii) a *record*[82], 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 syntax variant of the JSON syntax, where a list is are only syntactic sugar. It is also a weak syntax, in the sense where, given an input, the parser attempt to construct the best data structure, without complaining for lexical or syntactic errors, inferring punctuation like quote or comma, and so on. This definitely save edition time and also allows people not familiar with structured syntax to write symbolic data, without pain. This is indeed dangerous in the sense that it may generate a spurious data structure, but over time we have experiment that it is quite visible when rereading the data in a beautified format. The normalized string format is also more concise that the JSON counterpart format. Finally, any wJSON data is in one to one correspondence with the corresponding JSON format, thus avoiding yet another non-standard abstruse patch, whereas simply make JSON writing easier.

This is also a semantic variant called *wJSON*[83], 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

[80] https://fr.wikipedia.org/wiki/JavaScript_Object_Notation
[81] https://line.gitlabpages.inria.fr/aide-group/wjson/Value.html
[82] https://en.wikipedia.org/wiki/Record_(computer_science)
[83] https://line.gitlabpages.inria.fr/aide-group/wjson/#semantic

addition, two values are equal if and only, 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 ease web service or distributed computing "symboling" data exchanges, which could be done either in JSON or wJSON.

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 any semantic caveat, these are external mechanisms, and the obtained values are treated as new feature input.

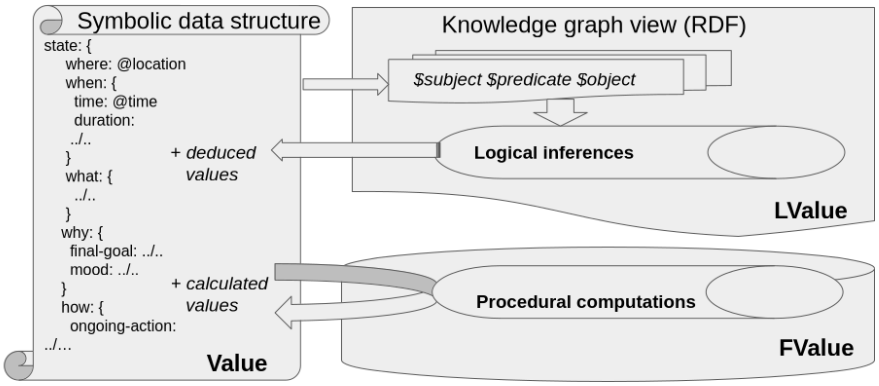


Fig. 22. Extended mechanism to compute calculated values using an external algorithmic function and deduced values using an external inference mechanism.

In Figure 22 example, a relation relates a data structure to another (using 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 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 some formula could be implemented. In particular a procedural method could interface with an external sensor or effector, for robotic application in the wide sense. At the *programming level*[84], we simply 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, this allows to obtain good computational properties and rather simple operational semantic. One key point is that such a straight-line piece of code does not generate infinite loops, and has a rather small polynomial calculation time, that can be numerically bounded. This correspond to a common organization of the code, where straight-line pieces of code are separated from non predictable loops or recursive part of the code.

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 [38] for an introduction in this context. Very simply, individuals have data-property corresponding to qualitative or quantitative feature property and object-property corresponding to relation. More precisely this corresponds to some *A-box*[85] of a knowledge graph.

Thanks to this we can add some ontology predicates, using preferentially *RDFS*[86] level description, if sufficiently powerful, in order to define a *T-box*[87] which will generates *deduced* features, say "LValue", as illustrated in Fig-

[84] https://line.gitlabpages.inria.fr/aide-group/wjson//FValue.html
[85] https://en.wikipedia.org/wiki/Abox
[86] https://en.wikipedia.org/wiki/RDF_Schema
[87] https://en.wikipedia.org/wiki/Tbox

ure 23. This requires a one-to-one transformation between a hierarchical structure and a flat relational graph: One solution, called "Turtoise" is rather obvious and already developed as *available here*[88]. Beyond RDFS, the *OWL*[89] description language or some related derivation rules using *SWRL*[90] 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 on the ontology language and are not part of the data system.

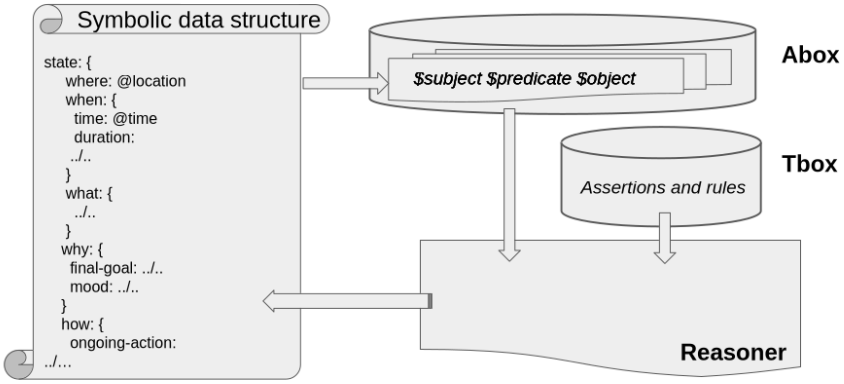


Fig. 23. Interface between the symbolic data management system and an ontology reasoner to derive feature values from inference. The input data corresponds to the A-box: the facts. The inference rules are in the T-box.

When implementing *LValue*[91] with existing ontology reasoner's API in Java or C/C++, we have experimented some painful caveats: complex specifications, loss of performances in the interface, the worst being an opaque behavior. This is indeed due to the fact we attempt to use them without enough level of expertise at the tool usage level. We then had the chance to make use of a very interesting trick: We install the *protégé*[92] rather universal ontology software, which is primary to be used via a manual user interface. Using it allows to consider a wide number of reasoners, obtain a maximal feedback on what is loaded, and so on. This software is started by the program itself, which then uses *xdotool*[93] which let simulate keyboard input and mouse activity, move and resize windows, thus replacing user manual gesture by a script. The drawback is that we must run the LValue mechanism on a desktop, but since we still are at a preliminary experimentation level, this is more than useful. This trick may have other uses.

Appendix E. Using such symbolic approach in human problem solving

In order to illustrate another use of such editing distance illustrated in Figure 7, let us consider a creative open problem solving experiment observation, using a pedagogic robotic activity, called "CreaCube" and presented in [52]. This section highlights a few elements of [39] for the self completeness of the paper.

The activity involves the manipulation and assembly of modular robotic cubes to build a vehicle, as explicitized in Figure 24. It is ill-defined in the sense that no more information is given, so that the subject has to consider

[88] https://line.gitlabpages.inria.fr/aide-group/wjson/turtoise.pdf
[89] https://www.w3.org/TR/owl2-primer
[90] https://www.w3.org/Submission/SWRL
[91] https://line.gitlabpages.inria.fr/aide-group/wjson/LValue.html
[92] https://protege.stanford.edu/
[93] https://github.com/jordansissel/xdotool/blob/master/README.md

affordance[94] and use creativity to find out all elements of one of the several possible solutions. In our framework, problem-solving is viewed as trajectory generation from the initial state to one goal state. The fact the state space is equipped with a metric and geodesic paths is thus crucial to work with such geometric view. In this context, operators are limited by the learner's procedural knowledge and discovery of affordances. The goal could be loosely defined, thus potentially reached in multiple states.

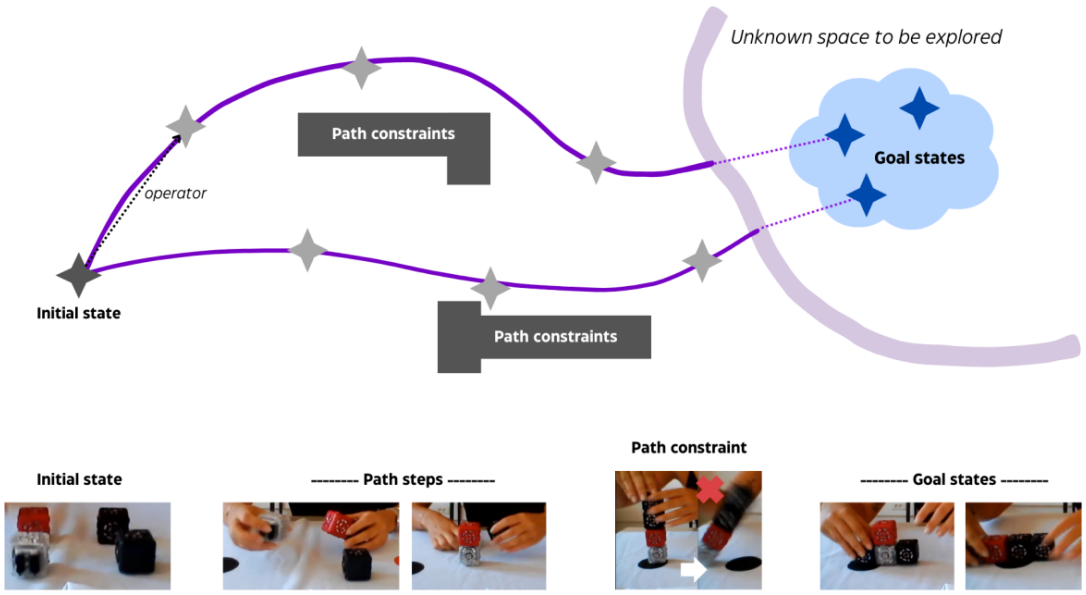


Fig. 24. Problem-solving as trajectory generation, in the CreaCube paradigm, from [39].

All experiment observables are specified as enumerations of possibilities, collected in a record in the sense defined in this paper. A temporal sequence of such states defined the activity representation from the subject initial situation the final success or give-up state, as trajectory. Clearly a suitable editing distance between each state of resolution allows to represent the subject progression, while the editing distance between two subject trajectories allows the authors to analyze resolution strategies. The Figure 25 describes a part of the system states to which is added, for example, the identification of each cube (recognizable by its color: dark blue battery, black sensor, white motor, red inverter) and the states of the cubes (e.g., "connected/disconnected" or "on/not on wheels"). This sub-ensemble of the possible states corresponds to the observables that have been chosen to analyze the activity.

This has been applied in [5] and [39]. It is shared here as an illustration.

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

[94]https://en.wikipedia.org/wiki/Affordance

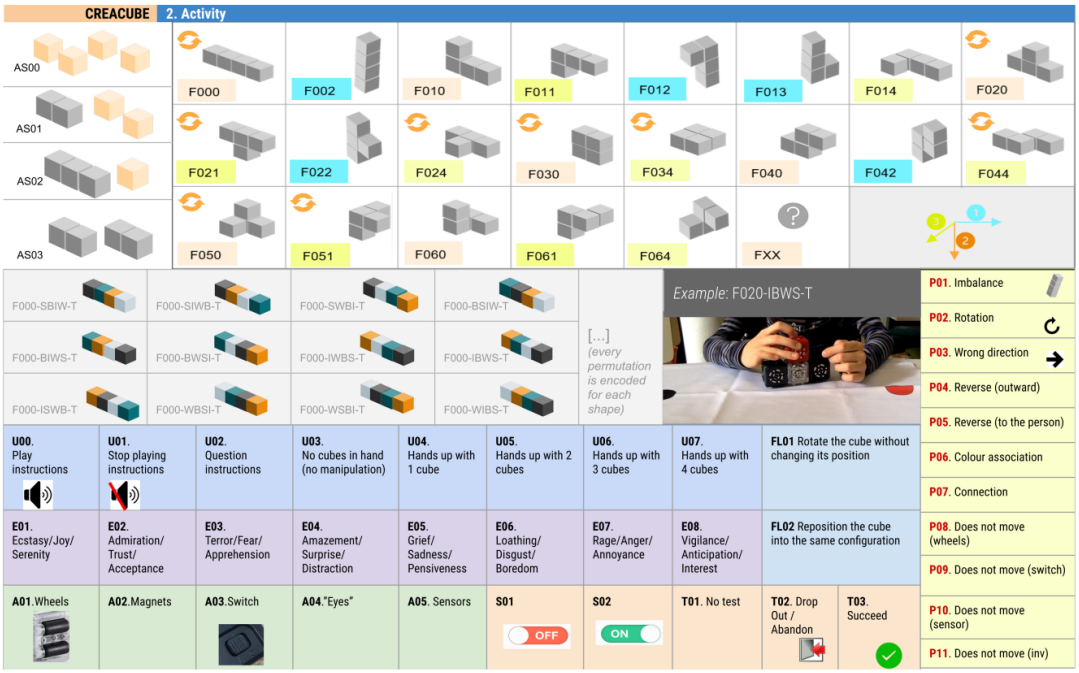


Fig. 25. The CreaCube observables description, from [39], after [**?**].

[5] J. Aubert, L. Köhler, C. Jozet-Alves, L. Lehéricy, P. Reynaud-Bouret and M. Romero, Estimating the learning behavior of an agent in a problem-solving experiment, 2024, Publisher: NeuroMod Institute Published: Annual meeting of the NeuroMod Institute. https://hal.science/hal-04711297.

[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] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

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

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

[28] 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.

[29] 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.

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

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

[32] 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.

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

[38] C. Mercier, L. Roux, M. Romero, F. Alexandre and T. Vieville, Formalizing Problem Solving in Computational Thinking : an Ontology approach, in: *2021 IEEE International Conference on Development and Learning (ICDL)*, IEEE, Beijing, China / Virtual, 2021, pp. 1–8. ISBN 978-1-72816-242-3. https://inria.hal.science/hal-03324136.

[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, H. Chateau-Laurent, F. Alexandre and T. Viéville, Ontology as neuronal-space manifold: towards symbolic and numerical artificial embedding, in: *KRHCAI-21@KR2021*, Hanoi, Vietnam / Virtual, 2021. https://inria.hal.science/hal-03360307.

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

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

[46] 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.
[47] 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.
[48] 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.
[49] 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.
[50] 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.
[51] 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.
[52] 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.
[53] M. Romero, A. Palaude, C. Mercier, T. Vieville and F. Alexandre, Deciphering Cognitive Processes in Creative Problem Solving: A Computational Approach, 2024, submitted.
[54] A.L. Rusawuk, Possibility and Necessity: An Introduction to Modality, 2018. https://1000wordphilosophy.com/2018/12/08/possibility-and-necessity-an-introduction-to-modality/.
[55] 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.
[56] 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.
[57] 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.
[58] 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.
[59] 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).
[60] 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.
[61] 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.
[62] 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.
[63] T. Viéville, An improved biologically plausible trajectory generator, Research report, RR-4539, Inria, 2002. https://hal.inria.fr/inria-00072049.
[64] 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.
[65] 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.