Metrizable symbolic data structure for ill-defined problem solving

Axel Palaude, Chloé Mercier, Thierry Viéville

Metrizable symbolic data structure for ill-defined problem solving

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

Submitted to Neurosymbolic Artificial Intelligence

Abstract. Within the context of cognitive computational neuroscience modeling, with both symbolic and numerical approaches, i.e., neurosymbolic artificial intelligence, we propose here an unusual way to relate explicit and readable data structure, with biologically plausible numeric operations: Very simply, 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. We also define in this symbolic space the notion of data region (i.e., data type or concept) and the concept of 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, providing that the data structure distances have been properly parameterized for a given application domain.

The proposed specification is not only developed formally but implemented in detail at the programming level, 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, with both symbolic and numerical approaches, i.e., neurosymbolic artificial intelligence, we propose here an unusual way to relate explicit and readable data structure representing both the system state and its related meta-data with biologically plausible numeric operations.

To this end, we need to first position the proposed technical development with respect to brain modeling at a symbolic level, both in terms of paradigm and in terms of knowledge representation, while also briefly reviewing existing approaches in order to position the originality of what is proposed here. This is going to be briefly reviewed in this section.

We must then define what we precisely mean by a “symbolic” approach for both the system state and its related meta-data, i.e., for both what is encoded in the neuronal ensembles and the modeling knowledge allowing us to

*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.

explain and interpret it. To this end, in the next section, we revisit what are signs and symbols, discussing the underlying concepts, and allowing us to share our design choices. We then detail a modeling approach which instantiates the previous general principles, allowing us to motivate the specification developed in the sequel.

Based on this, the third section describes at the computational level, how the previous data structure ingredients can be formalized, including at the programming level, allowing 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.

The final section discusses how rather generic machine learning algorithms can be applied at both the symbolic and numerical level simultaneously to encounter for generic mechanism of open-ended or ill-defined complex problem-solving, and reinforcement symbolic learning, and reports a little demonstration of the implementation mechanisms, before a short conclusion.

1.1. Brain modeling at a symbolic level

Studying the brain is performed at different temporal and spatial scales, different topological network scales, [7], but also at different "modeling" scales, in the sense of Marr, as reviewed and questioned in [30]. At the difference of Marr's original three implementations, algorithmic and computational levels, modern vision (see [27] for a comprehensive review) considers the implementation level, more precisely, as a biophysical representation in the wide sense, and considers as the highest level the cognitive behavior, while the computational stage, includes algorithmic aspects. This also allows us to clarify that the numerical, i.e., quantitative, aspects mainly stand on the biophysical side, while symbolic representation mainly stands on the behavioral stage, of course not exclusively, but more than a gradient. The key point is that computational representation must intrinsically be able to manage both representations conjointly, as illustrated in Fig. 1. This corresponds to the development proposed here.

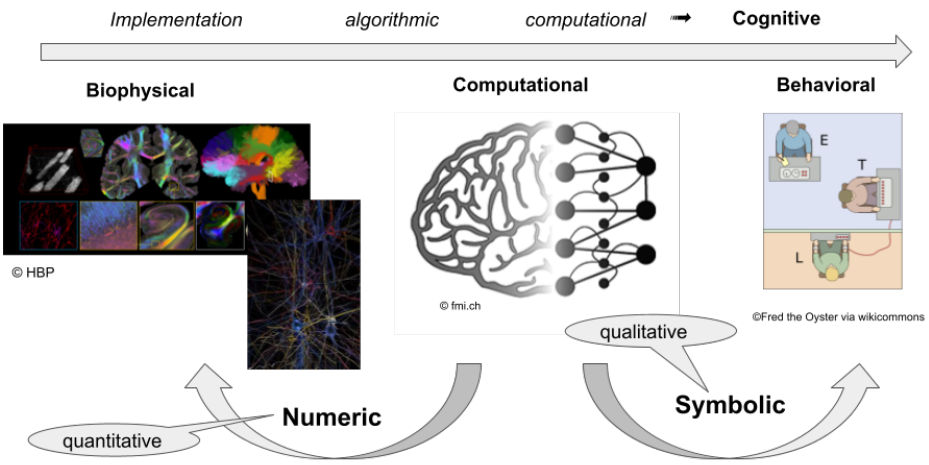


Fig. 1. A revised view of the Marr paradigm, considering modern development in computational neuroscience, see text for details.

From phenomenological to ontological brain modeling

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

model entities, each entity (e.g., a sensorimotor mechanism or a cognitive function) being defined via a key-word, a human-readable comment, properties and relations with other entities. This is more or less an informal ontology (see [13] for a general introduction and [42] for a practical approach). A step further, the brain is modeled using such a well-established framework, as explained in [26]. The brain anatomy has already been formalized for a rather long time, [20], while macroscopic cognitive models correspond to work in progress: problem-solving tasks have been addressed [23] focusing recently in computational thinking [35] when engaged in a creative open problem-solving activity [39]. These are external descriptions of the brain and it would be interesting to also introduce such symbolic data representation in models simulating the brain behavior itself, as discussed now.

1.2. Neurosymbolic approaches

Numeric machine learning mechanisms, i.e. calculating, 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, i.e. 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 [14]. These features are essential if such learning mechanisms are used to model cognitive functions. In addition, formal reasoning is by no means less costly than billions of operations of deep networks computation, as studied by [19], which limit their academic use for usual research teams, beyond raised ecologic and ethical issues, discussed by the authors. The use of such "black-box" tools at the modeling level is also questionable as analyzed in [49].

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 [25]. The present work precisely aims at considering coupling levels where symbolic knowledge and rules are compiled in distributed calculation either using localist 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 [12] 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. This may be another interest of the present approach.

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 [56] 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 basic but better-integrated level, such a plan is represented by the notion of abstract path, as developed in this paper. The link between symbolic and numeric approaches when considering domain-independent planners is also well illustrated in [4] and we wonder if our data representation could not be an interesting tool, for such an architecture. This work is also in link with ontology-related reasoning as, for instance in [32] for which the symbols are optimally encoded in the network numeric variables. Here as a dual approach, numeric elements cover symbolic data in our case.

Computational neuroscience models are mainly based on numeric mechanisms, except for instance VSA architecture (after [22]) based on Semantic Pointer Architecture (SPA) introduces an intermediate algebraic abstract view of spiking neural network architectures, in order to develop biologically plausible cognitive functionalities and human knowledge representations [15], including high-level symbolic representation allowing reasoning [40], regarding not only deductive but also inductive and abducting reasoning, as discussed in [36, 37].

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.

2. Symbolic data representation

2.1. Symbolic data representation using signs

What is a symbol? Apparently obvious, this notion is the ground of our human knowledge, and when daring to consider "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" in the sense of [31], as reviewed and discussed in [53], when it is semantically *grounded*, in the sense of the "symbol grounding problem", understood as the process of embedding symbolic computations onto real-valued features [5], thus providing a semantic interpretation or model (in the sense of a model of a set of logical assertions) of the symbolic system, which involves the capacity to pick referents of concepts and also a notion of consciousness. This includes affordance, i.e., not only features but also the capability of interaction with it, to attain an objective, and receive some outcome. This means that it is no longer an abstract set of assessments (potentially without any concrete implementation) but something that corresponds to a real object or behavior.

As discussed in, e.g., [45], concerning the emergence of symbolic thinking (see, e.g., [16] for a detailed discussion), the key problem is "ungrounding", i.e., how a symbolic representation can emerge from sensorimotor features and interaction with the environment. This aspect of the emergence of symbols, i.e., the fact that a symbolic representation emerges from a biological or any physical system in interaction with its environment, is enlightened by the semiotic approach as reviewed in [16], first considering a wider notion of "sign" and introducing a hierarchical meaning of an "icon" built only from sensorimotor features, i.e. at the level of the likeness with the object, e.g., with features such as color or smell, structures at an "index" level built by concrete relationships between given objects, i.e., at the level of a relation with the object, e.g., a weathercock indexing the wind direction and strength, thus giving rise to a "symbol" in the semiotic sense, which corresponds to abstract general relationships between concrete concepts or sensorimotor features, but with a qualitative break-up concerning concrete object features, e.g., a road sign, as schematized in Fig. 2 .

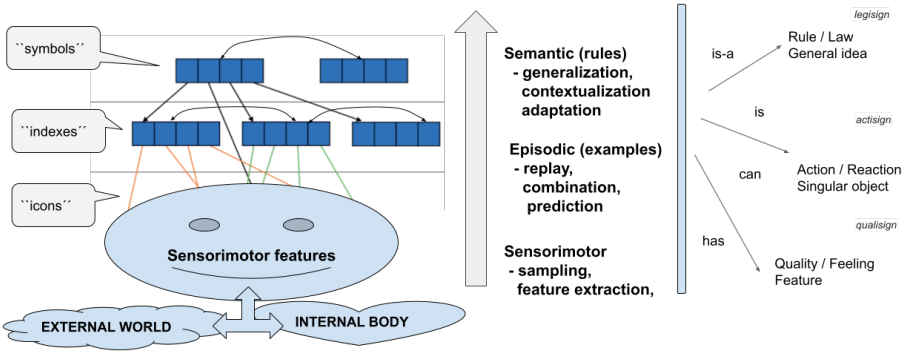


Fig. 2. The semiology hierarchy of signs and symbols, see text for details.

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

Given the previous considerations to share the context and motivate such a symbolic approach, let us now present which kind of representations is targeted here, following [36] modeling approach.

The aim is to manipulate the internal symbolic representation of knowledge of the form shown in Fig. 3, as introduced in [37]. Concepts are represented as a hierarchical data structure, in the sense of, e.g., [21], as a complement to associative and sequential memorization, identified in cognition. Concepts are anchored in an input/output, i.e., stimulus/response, framework, which might consist of sensorimotor feature spaces (colored regions) corresponding, for example, to different sensor modalities. Inherited features (e.g., 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., a penguin can also swim but 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 that is not explicitly manipulated by the agent but is used for process specification or interpretation (e.g., the weight unit and bounds).

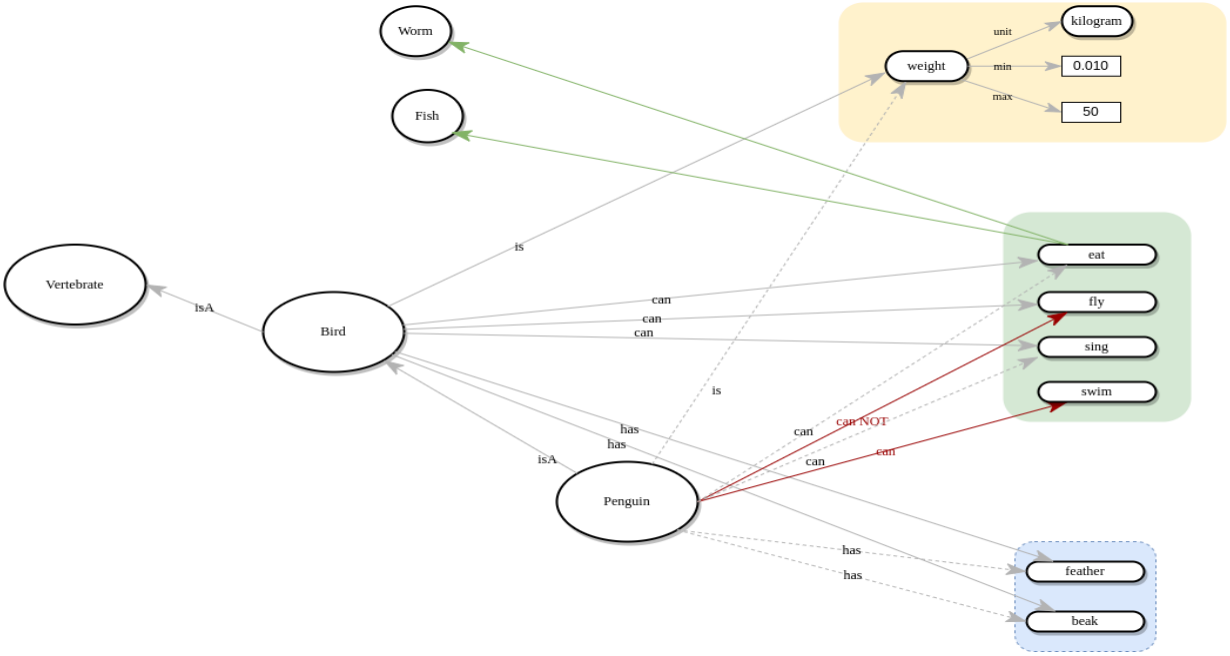


Fig. 3. Hierarchical data structure representing concepts, see text. From [37].

This corresponds to the [28] approach, with the simple idea that an individual resource can be defined by "feature dimensions," i.e., attributes with some typed value. For instance, a bird could be the following. The used syntax is a weak form of the JSON syntax.

```
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 }
}.
```

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`), is a typical sufficiently general [34]. This illustrative example is sufficient to allow us to detail the main characteristics of our representation. 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.,

{`sing`, `fly`}), or quantitative (e.g., the `weight`). The features can be hierarchical, either because the value is an enumeration (e.g., `can`) or because the value has some features (e.g., `weight`).

Such a data structure defines a "concept" in the sense of [28] (e.g., "a bird"), which is both a convex region of the state space (e.g., the region of all birds) and a prototype: Each feature has a default value, and this also defines a prototype (e.g., a typical, i.e., prototypical, bird). It corresponds to the third cognitive memory architecture, as proposed by [21]. At the programming level, it is going to be implemented as a "type". At the geometric level, data value corresponds to points and concepts to regions, but with a tricky property: Any data structure is the prototype of a region, as detailed in the next section.

When defining such data structure, there are several design choices and the following general recommendation might be useful:
- *Atomic value*: It is always better to decompose the information as much as possible in atomic irreducible elements (e.g., `family_name: Smith first_names: [John Adam]` instead of `name: 'Smith, John Adam'`) for algorithmic processing.
- *Maximal tree structure*: It is always better to organize features in sub-structures than to present flattened information (e.g., 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 A.3 for a discussion at the numeric data representation level.
We also point out, at the very concrete implementation level, that it is always preferable to choose explicit and standard names for features, i.e., look at already established vocabulary, otherwise avoid acronym or abbreviation, i.e., choose the most common word for the feature to name.

Using Vector Symbolic Architecture implemented at the neural spiking assembly level thanks to the Neural Engineering Framework [22], such a cognitive symbolic data structure can be implemented as biologically plausible memory, allowing to manipulate it conjointly at both a symbolic and numeric level [37].

3. Symbolic data specification

3.1. 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, i.e., the fact that 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 both a well-defined distance and geodesics, i.e. a minimal distance path from the initial value to the target value. The key point is that such distance depends on the data type[1].

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 with enriched meta-values.

string	This data type specifies a subset of strings.
modal	This data type specifies a level of truth between -1 (false), 0 (unknown), and 1 (true), see Appendix A.4.
numeric	This data type specifies a numeric value with its related metadata (bounds, precision, unit, ...), see Appendix A.3.

On the other hand, compound types correspond to usual enumeration, ordered lists, unordered sets, or named tuples, i.e., records.

[1] For instance, given a numeric value, it will depend on the chosen bounds, scale, and precision, as developed in Appendix A.3.

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

Specification details and type parameterization are detailed in Appendix A.

The present preliminary implementation only considers the basic atomic type, but it would be very easy to include all usual data types, used for instance in OWL.

Furthermore, structured data types such as (i) date and or time, (ii) IRI including URI, thus URL, (iii) geolocation, (iv) human language tags and locale, there is a one to one mapping between a string with specified syntax, and the data record items. The RecordType implementation provides an elementary mechanism of parsing based on regular expressions while existing usual middle-ware manages more sophisticated data.

New types can be defined combining these ingredients, given specific parameters or deriving new types.

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.

3.2. Defining an editing distance

A resource is a forest[2] of semi-ordered tree [43] data-structure. Furthermore, the features of a given resource are semi-ordered (i.e., some are comparable but not necessarily all of them).

In detail, the editing distance is defined considering *editing operations* each with a positive cost. Such user-defined costs allowing to model the data space taking a priori application knowledge into account: we can weigh such costs to quantify the importance of a given feature.

The distance is well-defined as the sequence of editing operations with a minimal cost (see for instance [1] where several alternatives are also proposed and compared), and 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. This mechanism not only allows the definition of a distance between two inputs (as the minimal cost of editing sequences transforming one input into another), but also makes explicit which node has been added, deleted, or changed. In other words, it allows interpolating intermediate data structures between the two.

An important point is that an editing sequence is itself easily represented as a data structure, namely an ordered list of actions. It appears that this editing distance path is a geodesic as discussed in the next section. With this notion of distance and geodesic, the symbolic data space is a kind of "non-differentiable manifold" (in an informal sense).

Another important point is that we preserve the data structure type[3]. This is illustrated in Fig. 4.

For compounded data type, we consider *editing operations* [4] given an input (l+) adding, (l-) deleting or (l#) changing a value in a list, (t+) defining, (t-) undefining or (t#) changing a value in a tuple, each of these operations having a user-defined positive cost, related to the extended semi-distances. We thus restrict editing distances by preserving the tree filiation, computable in polynomial time[5] [43], which would not have been the case otherwise, or if considering the tree as a general graph or ontology portion.

At the algorithmic level, it appears that there are algorithms to effectively compute the editing distance of a given data type and make explicit the sequence of editing operations, as detailed in Appendix A, and summarized in Fig. 4.

[2] A set of disjoint tree data structure https://en.wikipedia.org/wiki/Tree_(data_structure).

[3] For instance a list can not become a set. If two data are of different non-intersecting types, complete deletion, and insertion is the only admissible solution.

[4] E.g., when changing a feature value, or adding, deleting or changing a value in a list, or defining, undefining or changing a value in a record.

[5] In general, considering the editing distance in a tree as a general graph, such as an ontology portion, is NP-hard, thus intractable [8] and sub-optimal techniques must always be considered [9]. However, considering restricted editing distances preserving the tree filiation, are computable in polynomial time [43], and we propose a variant of this approach here.

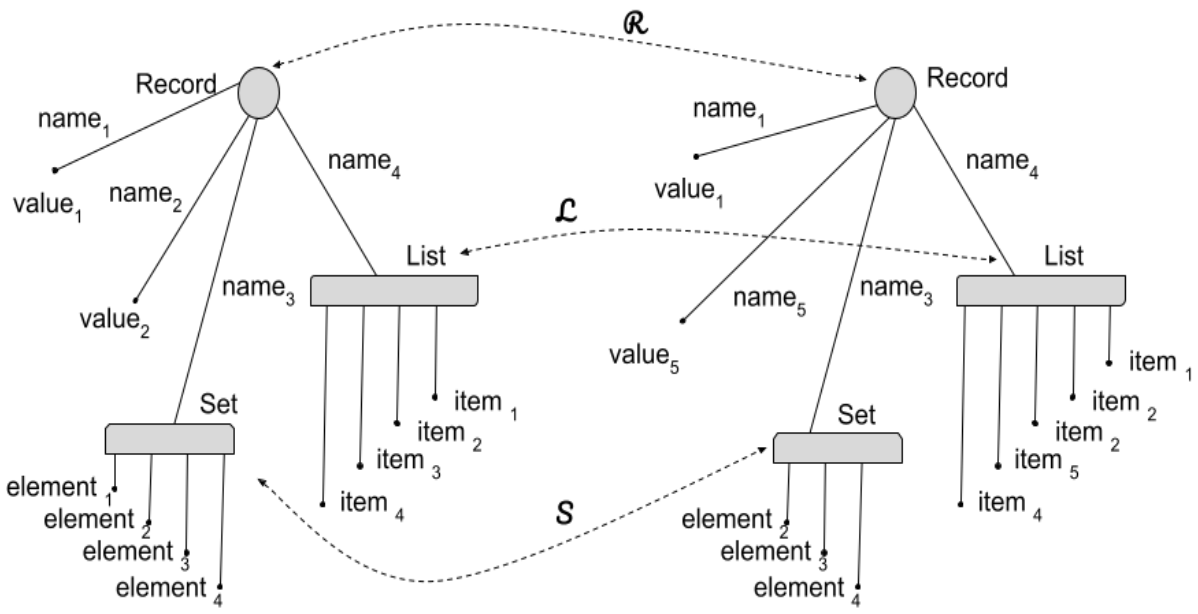


Fig. 4. Illustrating the notion of edit distance, for the main structured data types.
For the record, each named value is linearly (i.e., in $O(N)$) treated one by one, since it corresponds to distinct type values: Here the name_1 is unchanged, thus at a zero distance, the name_2 value is to be deleted, i.e. edit to correspond to an empty value, the name_5 value is to inserted, i.e. edit from an empty 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 (i.e., in $O(N^2)$) algorithm is used [41]: Here an insertion, the second item_2, and a modification of item_3 to item_5, allows to transform one list to another, while 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 (i.e., in $O(N^3)$) algorithm is used [10]: Here the deletion of element_1 makes the job.

3.3. Local structuration 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 this path, by construction of the editing distance[6]:

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

[6]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 $\mathbf{s}_2$, i.e., 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.

which is the reason we call it a geodesic.

A key point is that the path is not unique: in our case, at the implementation level we have made the following choices:

- For lists, i.e., sequences, at equal cost we prefer edition to deletion, and deletion to insertion, i.e., we maintain, else reduce the list length than increase it.
- For sets, i.e., unordered list, we first consider the association of maximal cost before the association of lower costs.
- For records, i.e. labeled tuples, we consider the transformation in the item order, i.e. modifying the 1st item 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, and the semantic level, i.e., when meaning is given the different elements has an impact on the result. Interestingly enough, this is easily adaptable.

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.

At a more theoretical level, we see here the split concerning 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 pointy, 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. Though not useful for our developments, the theoretical study of such a metric structure would be of some further interest.

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.

3.4. Using data type to specify concepts

A region of the symbolic state space corresponds to a data type. At the modeling level this corresponds to a concept, in the sense defined before and introduced by [28], and the implementation design choices, detailed in Appendix A, allow that such a region is convex in the sense that along a geodesic between two values of a given, all values are of the same type. It is also rather straightforward to verify that this forms a compact, complete, and path-connected metric space[7].

The proposed representation can also borrow from the usual data structure representation the notion of "schema". A schema defines a set of resources that verifies some constraints, e.g., for which a feature is defined or not, or for which quality is of a given data type[8]. If a given data structure is compliant, its distance to the defined set is zero. Otherwise, at the programming level, human-readable messages are shared to explain what is wrong.

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 defines a subset of compliant data structures. If a given data structure is compliant, its distance to the defined set 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. The development[9] such a projector 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

[7] Regarding connectedness, there is an important qualitative property. Given two data structures the editing distance is bounded since to transform one to another, we always can delete all qualities of the former and insert all qualities of the latter. If the minimal editing distance equals this maximal bound, it means that both resources have nothing in common, i.e., that 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, e.g., that modifying a value is always less costly than deleting and inserting it, this semantic connectedness defines a partition of the resource space.

[8] This corresponds to well-established XML-Schema https://en.wikipedia.org/wiki/XML_Schema_(W3C), or JSON-Schema https://en.wikipedia.org/wiki/JSON#Metadata_and_schema.

[9] For instance, in the schema each required quality has a default value, so that if missing, an insert operation can add it. If a numerical value is out of bound, the projection on the closest bound provides a straightforward solution. If the feature is of the wrong type, the only solution is to delete the value and insert a default value instead. And so on.

at the algorithm level, and it has to be specified for each data type. This is not always the case and that makes the difference between semantic and syntactic aspects, as explained in Appendix A. 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.

To define such a projector, each type has a default value, so that if missing, an insert operation can add it[10]. This induces the notion of a prototype of a type, as already mentioned.

A step further, as for the schema approach, the specification of type's parameters (e.g., numerical bound and precision, or list minimal length, ...) is itself a data structure, for which a vocabulary is defined, allowing to manipulate it a meta-level.

3.5. 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 "creative" solutions (in the wide sense).

The caveat is that we do not have a priori such a mechanism[11]. We thus have to design this new functionality, given the existing ingredients. Furthermore, as exemplified in footnote[11], when trying to simply generate unconstrained random values around a current value, by some simple random draw, given complex symbolic data structure there is almost no chance to generate a useful or even meaningful random value.

Using the data type bounds

We will get around with the idea of choosing some alternative "escape" target $\mathbf{s}_i$ to find on the path towards these targets a data structure away from the reference. By construction, the data space is bounded as soon as sufficiently specified:

string	Bounded if defined by either an enumeration or by a bounded[13] regular expression or a *maxLenth* meta-value bound.
modal	Bounded in $[-1, 1]$.
numeric	Bounded as soon as the *min* and *max* meta-value are defined.
enum	Bounded by construction.
list-of-*	Bounded as soon as a *maxLength* meta-value is defined.
set-of-*	Bounded as soon as a *maxLength* meta-value is defined.
record	Bounded as soon as the record items to take into count are explicitly enumerated.
value	Bounded by the fact that it is only the undefined value as explicit value.

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

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

[10] A step further, if a numerical value is out of bound, the projection on the closest bound provides a straightforward solution. If a feature is of the wrong type, the only solution is to delete the value and insert a default value instead. And so on.

[11] Given the standard Levenshtein editing distance[12] the editing operation have no well-defined "inverse". Let us consider a trivial example between `mama` and `mamie` easily obtained via, e.g., `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.

bounds. To consider a manageable number of bounds we easily can easily[14] only select a random subset of bounds.

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 Fig. 5. Choosing the best extrapolation path is an under-determined problem. Let us discuss how to better specify it.

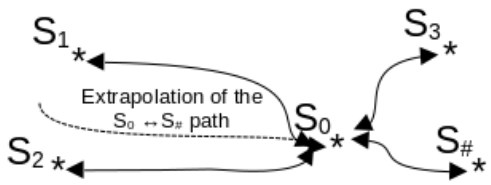


Fig. 5. Describing the extrapolation problem, see text for details.

Obviously the problem is meaningful only if $d(\mathbf{s}_0, \mathbf{s}_\sharp) > 0$; otherwise any value different from $\mathbf{s}_\sharp$ is a kind of extrapolation.

An interesting criterion, at the geometric level, is that geodesic from $\mathbf{s}_\dagger$ 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:

$$\nu(\mathbf{s}_\dagger) \stackrel{\text{def}}{=} d(\mathbf{s}_\dagger, \mathbf{s}_0) + d(\mathbf{s}_0, \mathbf{s}_\sharp) - d(\mathbf{s}_\dagger, \mathbf{s}_\sharp) = 0$$

as discussed previously, while $\nu(\mathbf{s}_\dagger) \geqslant 0$ in the general case, due to the distance triangular inequality. We thus have some advantage to minimize $\nu(\mathbf{s}_\dagger)$, i.e., find a data structure on a geodesic from $\mathbf{s}_0$ to $\mathbf{s}_i$ which minimize $\nu(\mathbf{s}_\dagger)$. We also can hypothesize that $\nu(\mathbf{s}_\dagger)$ is expected to increase along the geodesic from the current value towards a bound: this is experimentally verified at the implementation level.

The second aspect is that we want the extrapolation to be as "away" as possible, i.e. that $d(\mathbf{s}_0, \mathbf{s}_\sharp)$ must be as high as possible if it allows defining a prolongation of the $\mathbf{s}_0$ to $\mathbf{s}_\sharp$ geodesic, whereas these are only valid locally. We thus have to bound $d(\mathbf{s}_0, \mathbf{s}_\sharp) \leqslant d_{\text{local}}$ in order to remain in a local neighborhood.

Given this constraint and specification, we now have a well-defined algorithmic definition: *given all escape data structures, find on each related geodesic the data structure of maximal distance, but below d_{local}, that minimizes $\nu(\mathbf{s}_\dagger) \leqslant 0$.*

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 two criteria,

[14]The implementation available at https://gitlab.inria.fr/line/aide-group/symbolingtype/-/tree/master/src 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.

One on the distance, selecting only target $\mathbf{s}_\dagger$ with[15]:

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

i.e., 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[16]:

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

e.g., $\mathbf{s}_1$ or $\mathbf{s}_2$ but not $\mathbf{s}_3$ in the figure, 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.

4. Defining Scalar fields of symbolic data structures

4.1. 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, e.g., a cost value, or a weight, etc. Such an ingredient is introduced and presented here.

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

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

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

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

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

4.2. Interpolation mechanism in a neighborhood

Given a data structure $\mathbf{s}_0$ for which the scalar value has not been fixed, we need to interpolate this value given known values in a neighborhood.

Given the fact we only have a metric space, i.e. distances, we define such a neighborhood which two parameters:
- Its cardinality K, i.e., we consider at most the K closest data structure.
- A maximal distance $d_{\max}$, i.e., 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 (see, e.g., [11]) or graph manipulation (see, e.g. [44]).

Given such a neighborhood, given the available tools, we are left with the following choices:
- If $K = 0$ the value is undefined.

---

[15] 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 $d(\mathbf{v}_k, \mathbf{v}_\sharp) > d(\mathbf{v}_0, \mathbf{v}_\sharp)$ and we easily verify that on the reverse 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 for $\mathbf{v}_k$ close to $\mathbf{v}_0$ the data structure metric space is regular enough for this property to be verified.

[16] 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, as:

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

while we still consider that this property is locally valid beyond the Euclidean case.

- If $K = 1$ we only can approximate the data structure scalar value by considering the closest known predefined value, i.e., the value of the data structure which is in this neighborhood singleton.
- If $K > 1$, we propose to iteratively reduce the neighborhood to a singleton, i.e., until $K = 1$. 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 Fig. 6.

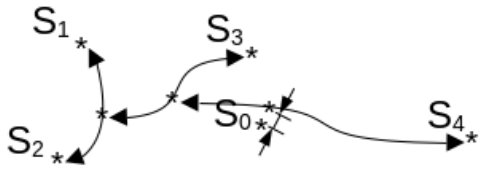


Fig. 6. As graphically represented here, $\mathbf{s}_1$ and $\mathbf{s}_2$ are replaced by the value on their geodesic as closed as possible to $\mathbf{s}_0$, which is then related to $\mathbf{s}_3$ by a geodesic, making the same choice again, which finally is related to $\mathbf{s}_4$, yielding to 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[17]

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

.

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[18].

Of course, this also applies to extrapolation since it is nothing but an interpolation with respect to bounds.

Although we can not explicitly consider that the data structures live in an abstract manifold, i.e., a space that is locally Euclidean up to the first order, we propose a method inspired by such formalism.

This method is to 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[19] The advantage of the former method is that it is more local, i.e., we perform linear approximation only between pairs of data structures that are as close as possible to each other, while this latter straightforward formula interpolates more distant values.

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

---

[17] This value is well defined as soon as:

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

the denominator being higher than 0.

[18] At this stage, our method is sub-optimal since we only consider one geodesic and not all possibles geodesics, and 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.

[19] For instance: $\nu(d) \stackrel{\text{def}}{=} e^{-\frac{d}{d_{\max}}} \simeq 1 - \frac{d}{d_{\max}}$.

4.3. Computing the barycenter of values

Given a set or subset of weighted data structures $\{(\mathbf{s}_1, w_1), (\mathbf{s}_2, w_2), \cdots\}$ we would like to approximate a data structure $\mathbf{s}_0$, such that[20]:

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

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

If the weights $\{\cdots w_i \cdots\}$ corresponds to the value $\{\cdots r_i \cdots\}$ we obtain the weighted centroids of the predefined set values.

If $K = 1$, obviously, $\mathbf{s}_0 = \mathbf{s}_1$ is the solution since we have a minimum when $d(\mathbf{s}_0, \mathbf{s}_1) = 0$.

If $K = 2$, it is a reasonable choice to minimize $l = w_1\, d(\mathbf{s}_0, \mathbf{s}_1)^e + w_2\, d(\mathbf{s}_0, \mathbf{s}_2)^e$ 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[21]. This is easily performed by scanning the different data structures to find the minimal value, when not simpler[22].

For $K > 2$, as before we can only work on geodesics and we propose to consider the two data structures $\mathbf{s}_i$ and $\mathbf{s}_j$ with the smallest distance, calculate on their geodesic the barycenter $\mathbf{s}_\bullet$ with weight $w_\bullet = w_i + w_j$, replacing the two former data structures by the latter, thus decreasing the set of data structure cardinal by 1, this being iterated until $K = 2$. This is not an optimal strategy, since we only consider a linear number of geodesics and not all possible ones, but this provides an approximate estimation, by considering only local operations.

5. Discussion and 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 Fig. 7, showing what is to be defined at the design level (Fig. 7.A) and what is derived thanks to this (Fig. 7.B). Moreover, as schematized in Fig. 7.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 [37].

These tools allow the application of usual machine learning algorithms, for instance, reinforcement symbolic learning [38].

5.1. Implementation and tiny demonstration

We have defined a metric space over a symbolic data structure that allows the implementation of generic machine-learning numeric algorithms directly on the data structure without requiring any approximate statistical embedding. Preliminary experiments have been conducted, for instance showing morphing[23] operations in the symbolic space [6] as illustrated in Fig. 8.

Let us finally discuss some applications.

---

[20] 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 (see e.g. https://en.wikipedia.org/wiki/Centroid), since the linear combination of vectors $\mathbf{v}_i$ is defined. However, this is not the case with data structures, where we only can consider distances.

[21] Let us consider a general data structure $\mathbf{s}$, writing $d_1 \stackrel{\text{def}}{=} d(\mathbf{s}, \mathbf{s}_1)$, $d_2 \stackrel{\text{def}}{=} d(\mathbf{s}, \mathbf{s}_2)$, $d_{12} = d(\mathbf{s}_1, \mathbf{s}_2)$, thus $d_1 + d_2 \geqslant d_{21}$, i.e., $d_1 + d_2 = d_{12} + \epsilon, \epsilon \geqslant 0$, we are now left to minimize $l = w_1\, d_1^e + w_2\, (d_{12} + \epsilon - d_1)^e$ and it is clear that this is minimal for $\epsilon = 0$.

[22] 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:

$$w_2\, d(\mathbf{s}_0, \mathbf{s}_1) = w_1\, d(\mathbf{s}_0, \mathbf{s}_2).$$

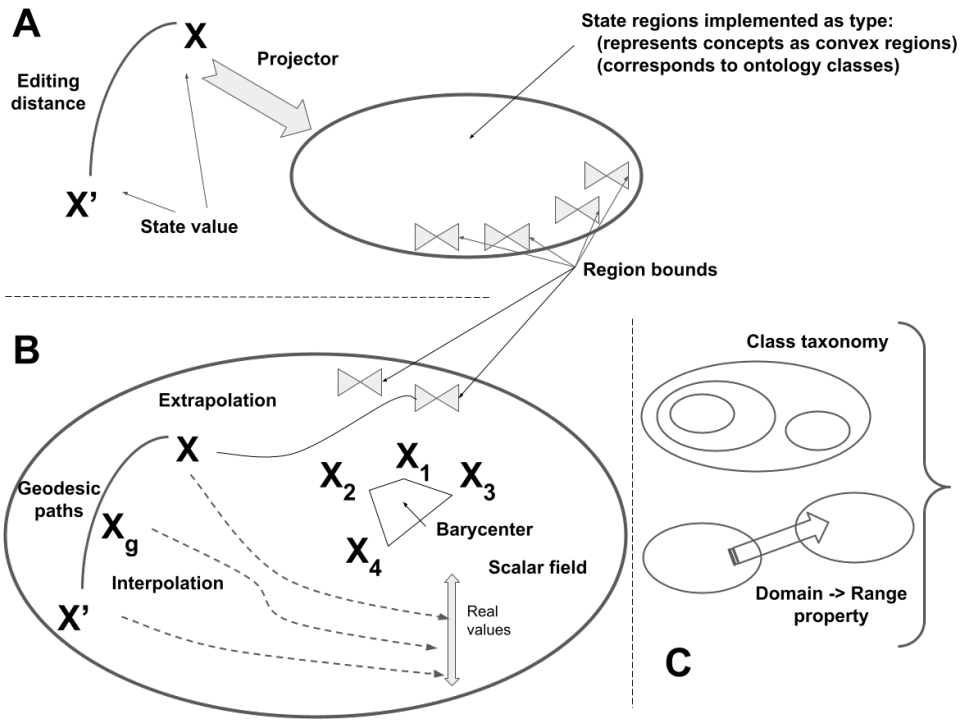


Fig. 7. Summarizing the different "symboling" tools:
A: Tools defined at the design level: the notion of a region corresponding to a type or concept, equipped with a projector, and equipped with a metric using an editing distance between the different state values, while bounds are to be specified.
B: 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 (see text).

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.

A step further, 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

[23] An image morphing demo example, using the SVG symbolic description of an image is available here:
https://line.gitlabpages.inria.fr/aide-group/symbolingtype/visualmorphing/titi-toto.html
and an audio morphing demo example, using the MIDI symbolic description of an image sound is available here:
https://line.gitlabpages.inria.fr/aide-group/symbolingtype/musicmorphing/au-clair-de-la-lune-frere-jacques.html
while the open-source implementation of SvgType and MidiType is available here:
https://line.gitlabpages.inria.fr/aide-group/symboling.

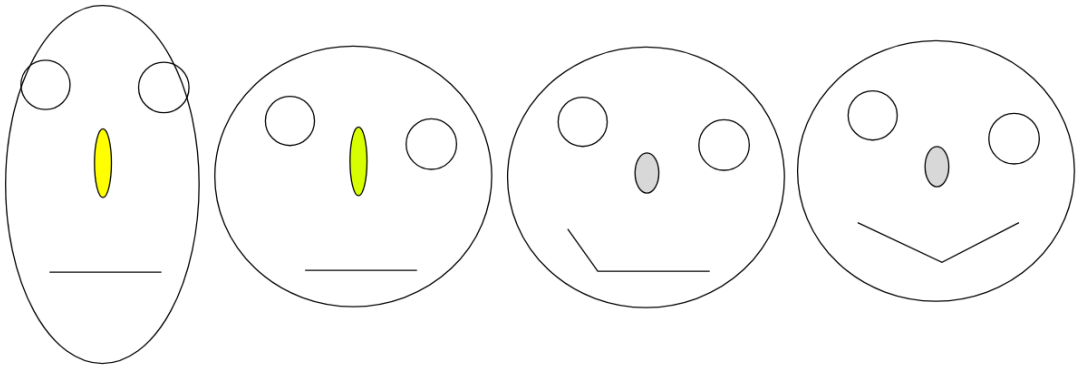


Fig. 8. A subset of the morphing between the “titi´´ drawing on the left and the “toto´´ drawing on the right: two intermediate images on the geodesic path defined by the SVG drawing representation are shown. This preliminary result shows how the symbolic data structure defining the drawing elements is changed, mainly from top to bottom, because it is the record item order. The intermediate color shows that the color space is also related to a distance in the SVG specification.

conceptual representation has to be adapted during the task execution, i.e., completed and corrected, as preliminary experimented in [46].

Application to creative problem solving

Creative problem solving requires divergent thinking as analyzed in [47] and formalized in [3]. In a nutshell, divergent thinking requires not only interpolating but extrapolating new resources from existing ones. Let us illustrate these opportunities by considering three non-exclusive examples, as illustrated in Fig 9:

Projective divergent extrapolation: Given a present state represented as a data structure, we may wish to extrapolate another state, only constrained by some requirement (e.g., we want to invent a "penguinemu" (a penguin morphing towards an emu)) which the only constraints that it has some of the 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 resource, by projecting the prototype onto the targeted schema constraints.

Sequential extrapolation: Let us consider two resources 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 [29], e.g. reasoning of the form “Robin is to Batman what Sancho is to Don Quixote´´ we can use the editing sequence from Robin to Batman in the source context, to re-apply it to Sancho in the target context in order to generate by analogy qualities that could apply to Don Quixote. The mapping from Robin to Sancho, i.e., from the source to the target domain, forms a commutative diagram with the source and target relations. This mechanism is iterative in [29] and performs only at a symbolic, namely ontology, level. Here we propose to implement it at our hybrid geometric level.

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 Fig. 10 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, i.e., a certain belief regarding what is stated, as formalized in Appendix A.4.

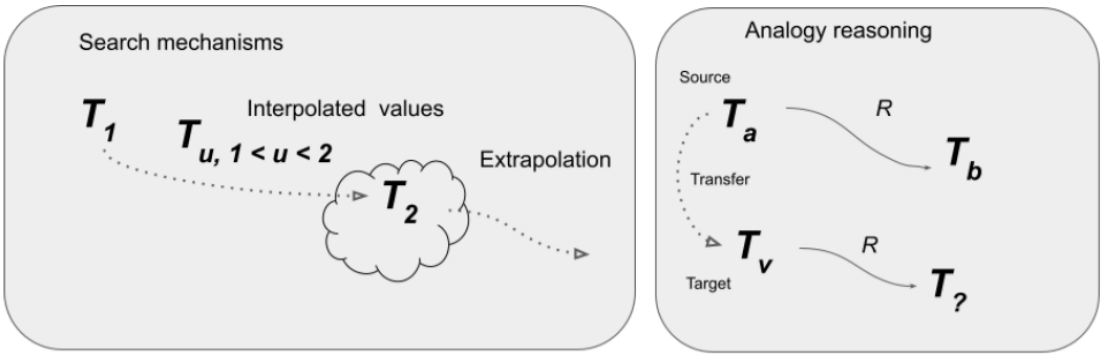
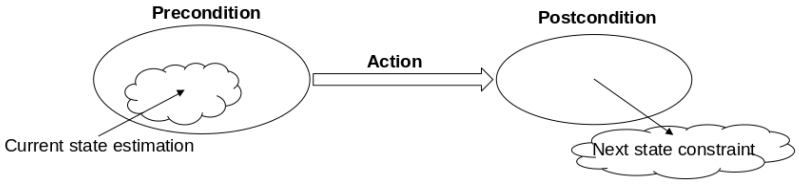


Fig. 9. A schematic representation of two kinds of generative processes. Left: Search, by extrapolation in the mirror of an interpolation between two structures. Right: Reasoning by analogy, as formalized in [29].




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

Fig. 10. An example of higher-level specification, to formalize rule-based behavior, as discussed in e.g. [2].

5.2. 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 [52], while [40] 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 [15] 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: [33] has 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 [51].

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.

5.3. Conclusion

Altogether, our design choice is to consider that any resource 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 resources, projection onto a resource 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 [38] for a preliminary version of the present framework.

5.4. Acknowledgments and contributions

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. References

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

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

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

[4] M. Asai, H. Kajino, A. Fukunaga and C. Muise, Classical Planning in Deep Latent Space, *Journal of Artificial Intelligence Research* **74** (2022), 1599–1686. https://jair.org/index.php/jair/article/view/13768.

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

[6] 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.

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

[8] 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.

[9] 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.

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

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

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

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

[26] 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.

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

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

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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..

[37] C. Mercier and T. Vieville, Where is Aristotle in our brain? On biologically plausible reasoning embedded in neuronal computation., *Cognitive Neurodynamics* **0** (2024), Under review.

[38] 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, Springer International Publishing, Bratislava, Slovakia / Virtual, 2021, pp. 608–612. ISBN 978-3-030-86380-7. https://inria.hal.science/hal-03327706.

[39] 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.

[40] 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.

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

[42] 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.

[43] 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.

[44] 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.

[45] 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.
[46] 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.
[47] M. Romero, A. Palaude, C. Mercier, T. Vieville and F. Alexandre, Deciphering Cognitive Processes in Creative Problem Solving: A Computational Approach, 2024, submitted.
[48] A.L. Rusawuk, Possibility and Necessity: An Introduction to Modality, 2018. https://1000wordphilosophy.com/2018/12/08/possibility-and-necessity-an-introduction-to-modality/.
[49] 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.
[50] 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.
[51] 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.
[52] 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.
[53] 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).
[54] 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.
[55] 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.
[56] 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.
[57] 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.
[58] T. Viéville, D. Lingrand and F. Gaspard, Implementing a multi-model estimation method, *International Journal of Computer Vision* **44** (2001), 41–64. https://hal.inria.fr/inria-00000172.

Appendix A. Symbolic data implementation

A.1. Semantic and syntactic aspect of implementation

Semantic specificity of the iterative data structure

At the implementation level, the data object model, the DOM, is a `Value` which is either (i) an atomic string (including string parsable as numeric or boolean value) or (ii) a record, i.e., an unordered set of elements accessed by label[24]. This forms an unbounded set of well-formed values.
In order to structure the data, the basic construction is the notion of `Type`. The set of values of a given type defines a metric subspace, i.e., a region of the state space equipped with (i) a distance between two values, and (ii) a projector from a neighborhood of this subspace onto it.
It appears that manipulating symbolic data structure requires specifying both a syntactic and semantic projection:

Semantic specificity of the iterative data structure

[24]This corresponds to a semantic variant of a JSON data structure, called wJSON https://line.gitlabpages.inria.fr/aide-group/wjson/#semantic, for which:
- Record name/value pairs are ordered, preserving the insertion order of record keys, or sorted in any application-related order.
- An array of syntax `[a b ...]` is equivalent and equal to a record `{ 0: a 1: b ...}` indexed by consecutive non negative integer.
- All atomic values cast from and onto string,
- The 'empty' value corresponds to an undefined default value, and more generally any type is expected to have a default value.

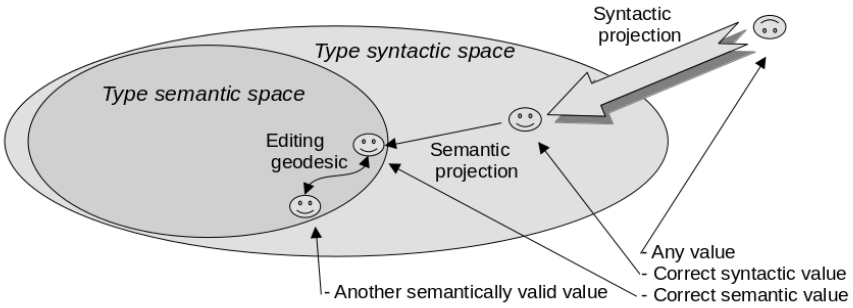


Fig. 11. Illustrating the notion of syntactic and semantic type, see text for detail.

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[25], such neighborhood is referred to as the type syntactic space[26].

Calculated or deduced feature values. Value can not only be input as data but can also be computed, as illustrated in Fig. 12.

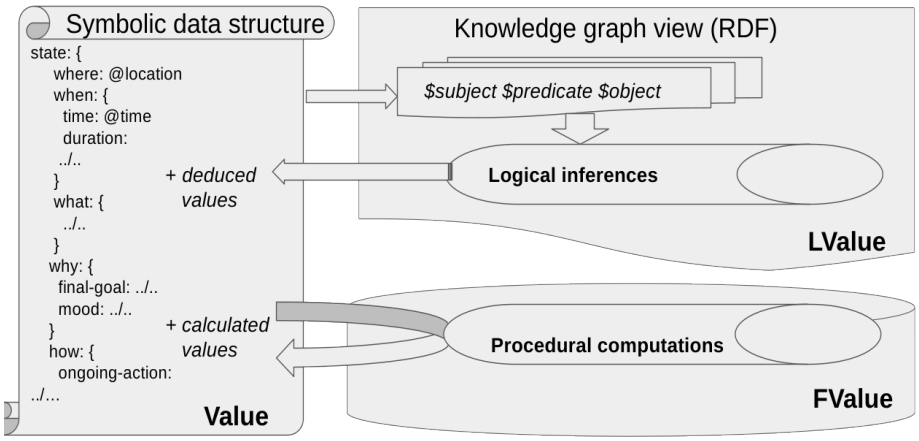


Fig. 12. Extended mechanism to compute calculated values using an external algorithmic function and deduced values using an external inference mechanism. To avoid any semantic caveat, these are external mechanisms, and the obtained values are treated as new feature input.

A relation relates a resource to another (e.g. `is-a-prey-for`) or allows to define some *class* (i.e., `is-a`). 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 (e.g., `has-capability` in the example). In other

[25]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 can not parse to a number, then any numeric operation will be undefined. If the string parses to a numeric value, it is easy to define how to project such a real number onto a positive integer.

[26]A step further, the syntactic neighborhood may correspond to a more general super type, value syntactically valid being semantically valid concerning this supertype. Thanks to this, the distance from a value to the type region can be calculated concerning the supertype metric, providing that the projection corresponds to the shortest distance.

words, some feature values, say "FValue", are calculated by algorithmic functions[27] hooked to the data structure, and generate *calculated* features. Depending on the application needs, further *calculated* features (e.g., calculating some numerical values using some formula) could be implemented.

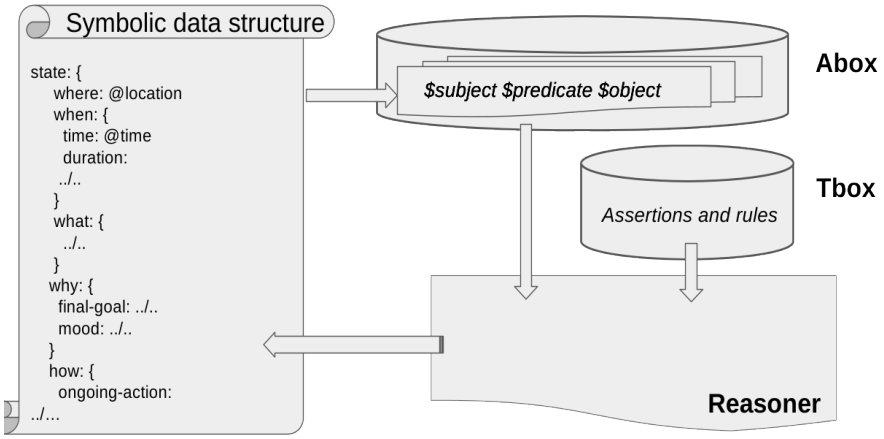


Fig. 13. Interface between the symbolic data management system and an ontology reasoner to derive feature values from inference. The input data corresponds to the A-box, i.e., the facts. In our setup, the inference rules, i.e., the T-box are defined directly on the ontology language and are not part of the system.

Another key point is that such representation is in one-to-one correspondence with what could be specified using an ontology approach (see, e.g. [35] for an introduction in this context) since 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 of a knowledge graph. Thanks to this we can add either some ontology predicates (e.g., using the RDFS or OWL description language) or some derivation rules (e.g., using SWRL rules) to define a https://en.wikipedia.org/wiki/Tbox which will generates *deduced* features, say "LValue", as illustrated in Fig. 13. This requires a one-to-one transformation between a hierarchical structure and a flat relational graph: The related specification, called "Turtoise" is rather obvious and already developed as available here.

A.2. Implementation of the basic data type

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.

[27] More precisely, we consider non-recursive imperative code constructs without side-effect (e.g., without global variables) made of elementary functions, sequences and tests, and enumerators of feature set, this allows to obtain good computational properties (e.g., small polynomial calculation time) and rather simple operational semantic.

– **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, i.e. 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 applied on the list element and delegating the edition cost to the item type distance computation [41].

  - 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 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.

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, 2016) implementation of the matrix version of the Hungarian algorithm, after [10].
- 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.

A.3. 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 (e.g., second, meter, etc), if any, as schematized in Fig. 14.

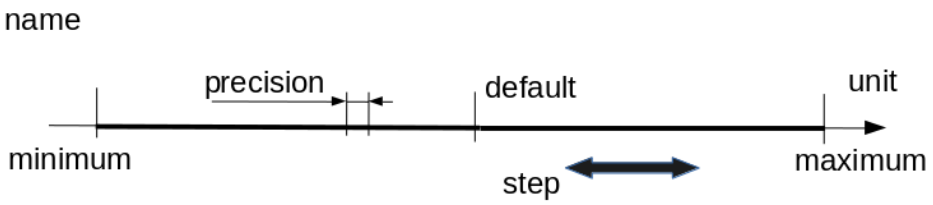


Fig. 14. Specification of a numerical value. Metadata includes a name, a unit, a default value, bounds (minimum and maximum), a precision under which two values are not distinguishable, and a step value corresponding to a neighborhood size used to cover the value range.

Such specification is important to properly manipulate quantitative information. In particular, the values can be normalized (e.g., mapped onto the $[-1, 1]$ interval) and 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, and so on (see [58] for a discussion). 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 numerical sensory data and internal quantitative data (e.g., derived data, calculation output, etc). A step further, symbolically coded data can always be sampled (e.g., a vector font drawn on a canvas and then sampled as pixels) at a given precision.

It is obvious that any quantitative measure is bounded (e.g. physical velocity magnitude stands between 0, for a still object, and the light speed) and is given up to a given precision (e.g., a localization in an image is given up to one-pixel size, a school ruler up to 1-millimeter graduation). The key point is that it is useful to make explicit this obvious fact (e.g., that any measurement device has a given precision and a measurement range) at the specification level instead of using it implicitly when required.

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 variational solver, i.e., optimizer and controller.

This specification induces a pseudometric:

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

in words, the distance is weighted by the *step*, i.e., the neighborhood approximate size, while if two 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.

A.4. 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 Fig. 15.

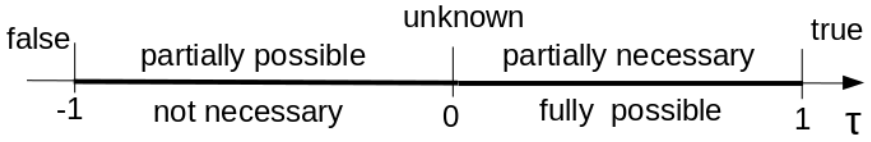


Fig. 15. Specification of a modal value of belief, this with regard to necessity and possibility, as defined in the possibility theory. The interpretation is that what is "not so false" is partially possible but not necessary and what is "partially true" is entirely possible but partially necessary. 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, i.e., impossible), and the unknown value to 0, which corresponds to a fully possible but absolutely not necessary value[28]. In between, negative values correspond to partially possible events and positive values to partially necessary events.

[28]This representation has been designed to be compatible with the ternary Kleene logic, and is in one to one correspondence with respect to the possibility theory. Possibility theory is devoted to 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 [17] 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

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.

theory, as discussed in [17] and [18], in link with modal logic, i.e., something true in a "given context" [24], which is also considered as representative to what is modeled in educational science and philosophy [48], because it corresponds to commonsense reasoning in the Piaget's sense [50], taking exceptions into account, i.e., considering non-monotonic reasoning. Furthermore, in symbolic artificial intelligence, i.e., knowledge representation through ontology, the link has been built between this necessity/possibility dual representation and ontology [54]. This must be understood as a deterministic theory, in the sense that partial knowledge is not represented by randomness. See [55, 57] for an extension taking randomness into account.