Symboling: Metrizable symbolic data structure

Position of the problem: Context and basic mechanism

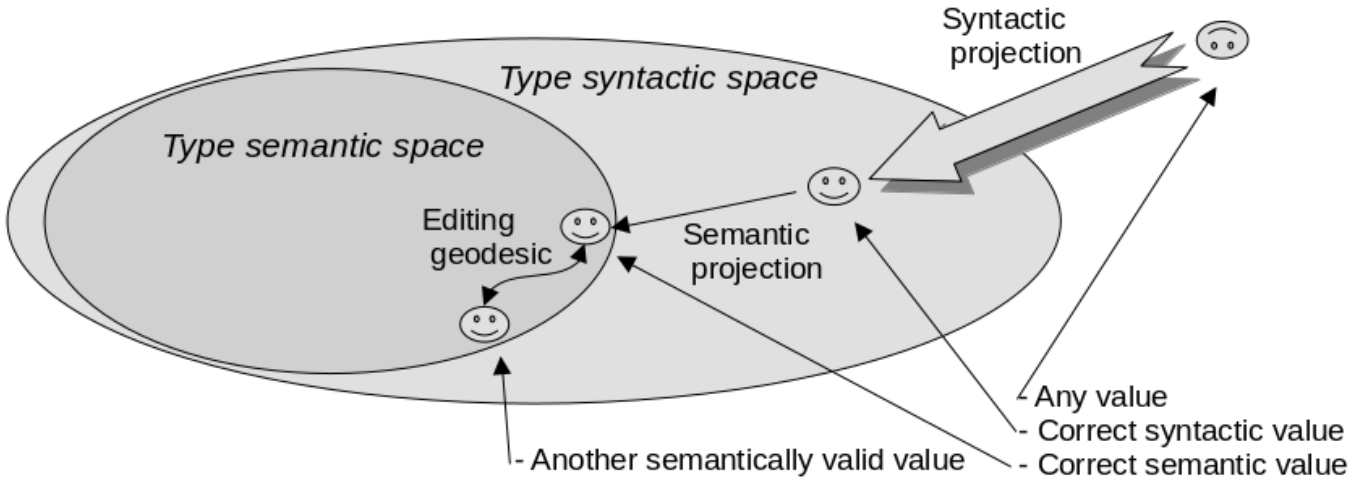
The basic ingredient is to generate a metrizable symbolic data structure embedding (say, a "symboling"). In the resulting metric space, the lever is the notion of editing distance, i.e., the fact that a symbolic data value is step by step edited in order to equal another value. Each elementary editing 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 projection of a data value in the neighborhood of the data type region onto it, as developed in [Palaude et al., 2023].

Implementation: Structuring via typed data value

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[2]. 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 to specify both a syntactic and semantic projection:

[1] For instance, given a numeric value, it will depends on the chosen bounds, scale and precision.

[2] This corresponds to a semantic variant of a JSON data structure, called wJSON, 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.



The key point is that in order 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[3], such neighborhood is referred as the type syntactic space[4].

Implementation: Basic data type

Thanks to the proposed design, the specification is modular and hierarchical:

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

[3]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 real number onto a positive integer.

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

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).
numeric	This data type specifies a numeric value with its related metadata (bounds, precision, unit, ...).

The present preliminary implementation only considers 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 a specified syntax, and the data record items. The RecordType implementation provides an elementary of mechanism of parsing based on regular expressions, while existing middleware manage more sophisticated data.

On the other hand, compound types correspond to usual enumeration, ordered list, unordered set, or named tuples, i.e., record.

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., a named tuples.
value	This data type is the root data type that corresponds to any value.

Specification details, and type parameterization is detailed in the source file documentation.

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

References

[Palaude et al., 2023] Palaude, A., Mercier, C., Kohler, L., and Vieville, T. (2023). Metrizable symbolic data structure for ill-defined problem solving. *Journal of Artificial Intelligence Research*, In preparation.