Taylor & Francis
Taylor & Francis Group

Science and Technology of Advanced Materials: Methods

ISSN: 2766-0400 (Online) Journal homepage: www.tandfonline.com/journals/tstm20

NIMS polymer database PoLyInfo (III): modularizing ShEx schemas for descriptors and properties in PoLyInfoRDF

Koichi Sakamoto & Masashi Ishii

To cite this article: Koichi Sakamoto & Masashi Ishii (2025) NIMS polymer database PoLyInfo (III): modularizing ShEx schemas for descriptors and properties in PoLyInfoRDF, Science and Technology of Advanced Materials: Methods, 5:1, 2544516, DOI: 10.1080/27660400.2025.2544516

To link to this article: https://doi.org/10.1080/27660400.2025.2544516

SCIENCE AND TECHNOLOGY OF ADVANCED MATERIALS: METHODS
2025, VOL. 5, NO. 1, 2544516
https://doi.org/10.1080/27660400.2025.2544516

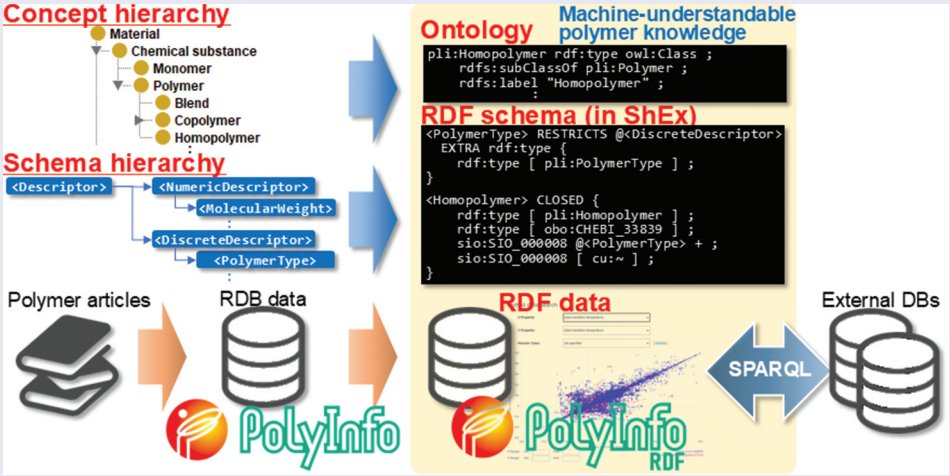
NIMS polymer database PoLyInfo (III): modularizing ShEx schemas for descriptors and properties in PoLyInfoRDF

Koichi Sakamoto and Masashi Ishii

Center for Basic Research on Materials, National Institute for Materials Science (NIMS), Tsukuba, Japan

ABSTRACT

PoLyInfo is a polymer database of the National Institute for Materials Science (NIMS) of Japan. In our previous work, to make the PoLyInfo data machine-readable and further machine-understandable, we built PoLyInfoRDF to store these data in the standard Resource Description Framework (RDF) format and then defined its schema in the Shape Expressions (ShEx) language. When designing the schema, it is important to modularize the schema such that the common components are reusable. This is the objective of this study and is essential for efficiently defining schemas of the descriptors and properties, which constitute the core of PoLyInfo, a large collection of experimentally measured polymer characteristics. As an example of modularization, descriptors of the source-based name and molecular formula both include a string value, hence their schemas may well share ('inherit') the schema for string values, which would be defined once and subsequently reused throughout the entire set of schemas. Actually we noticed a considerable amount of common portions among schemas of descriptors and properties, and clarified a 'schema hierarchy' to reflect the above 'inheritance' relationships, separately from the ontological 'concept hierarchy'. We then investigated the extent to which the adapted strategy was able to successfully define the PoLyInfoRDF schema. Under this schema hierarchy, inheritance mechanisms in ShEx played a significant role in sharing common portions effectively in a well-organized manner. We expect future developments based on our approach to contribute to the standardization of scientific data representation in RDF by providing a library of reusable schemas.



IMPACT STATEMENT

We have developed a new method for modularizing scientific data schemas and managing them hierarchically, and demonstrated it in PoLyInfo. This paves the way for data fusion in materials chemistry.

ARTICLE HISTORY
Received 9 April 2025
Revised 30 June 2025
Accepted 3 August 2025

KEYWORDS
Polymer chemistry; PoLyInfo; ontology; schema; RDF; ShEx; SPARQL; modularization; standardization

CONTACT Masashi Ishii ISHII.Masashi@nims.go.jp Center for Basic Research on Materials, National Institute for Materials Science (NIMS), Ibaraki, Tsukuba 305-0047, Japan

1. Introduction

1.1. Background

PoLyInfo is the polymer database of National Institute for Materials Science (NIMS) of Japan with more than half a million data points [1]. This is a relational database (RDB) for human users, equipped with a self-contained graphical user interface (GUI) to enable users to search for chemical information on polymers as guided through the user operations (PoLyInfo(I) [2]). Recently, ongoing advancements of data-driven approaches in this field, such as machine learning or materials exploration, also in terms of the PoLyInfo database, have boosted the demand for automatic remote access to the data collection from client machines according to a standard protocol. At the same time, a demand has arisen for programmatically processing the data for users' own purposes.

Against this background, we built (using the D2RQ tool [3]) PoLyInfoRDF, which is the Resource Description Framework (RDF) [4,5] version of PoLyInfo, to provide a machine-readable counterpart, where the data are stored in an open standard format and are ready for use through a web-accessible endpoint. This enables PoLyInfo to cooperate with other external public or proprietary triple stores and jointly contribute to federated data-driven research. Furthermore, we defined the PoLyInfoRDF schema [6] in the Shape Expressions (ShEx) language [7] (one of the widely accepted RDF schema description languages). This schema is a machine-readable formal specification of PoLyInfoRDF, and together with the PoLyInfo ontology [8] (a description of the vocabulary written in the Web Ontology Language (OWL) [9,10]), has made the PoLyInfo data machine-understandable to promote the mechanical processing of the data (PoLyInfo(II) [11]).

1.2. Subject matter

The overall design policies of the PoLyInfoRDF and its ShEx schema are described in PoLyInfo (II) [11]. Their origin, PoLyInfo, provides a wide variety of data such as the chemical structures of polymers, different names, and properties of polymer samples. When building PoLyInfoRDF, among these data, we paid particular attention to the properties (the chemical or physical behaviors of polymers, e.g. melting temperature and electric conductivity) as well as descriptors (values related to a particular aspect of interest, e.g. temperature values and voltage values). This is because they constitute the core of PoLyInfo, a large collection of observations on polymers for describing their various characteristics, which we think will play a substantial role in areas such as polymeric material design.

Furthermore, when designing the PoLyInfoRDF schema in ShEx, one of the major problems we had to overcome was to find ways to modularize the schema and reuse the common parts, which is essential for efficiently defining the schemas of descriptors and properties. For example, in the PoLyInfoRDF schema, descriptors of the same type, such as temperature descriptors, tend to appear many times. In such cases, the schemas for these descriptors should be defined once and referenced from other places (i.e. reusable) throughout the entire schema. Similarly, the schemas for most polymer properties share an identical portion. Again, the schema for this portion should be reusable among the various property schemas. We refer to this methodology as the 'modular approach', where in order to illuminate the common portions of schemas, we introduced the 'schema hierarchy', which is one of the key ideas in our approach and comes into play apart from the 'concept hierarchy' (or 'class hierarchy') demonstrated in the PoLyInfo ontology. In this paper (referred to as PoLyInfo(III)), we focus on defining the descriptors and properties in ShEx under modularization, where, taking advantage of the above schema hierarchy, inheritance mechanisms in ShEx play a significant role in sharing the common portions effectively in a well-organized manner.

1.3. Structure of this paper

The remainder of this paper is organized as follows. Section 2 provides a brief overview of RDF and ShEx, which we employed as frameworks for building PoLyInfoRDF and defining its schema. Section 3 examines the RDF graphs for descriptors and properties, that are the core of PoLyInfo data for describing various polymer characteristics. Section 4 reveals the contents of the PoLyInfoRDF schema, focusing on the definition of the schemas for descriptors and properties under our modular approach based on the schema hierarchy, which is the main theme of this study. Section 5 discusses the evaluation of our efforts, including the possible applications of ShEx schemas, such as assisting users with writing the SPARQL Protocol and RDF Query Language (SPARQL) [12] queries for PoLyInfoRDF, and the future potential of our approach from the viewpoint of standardizing scientific data representation in RDF. Finally, section 6 concludes the paper.

2. Overviews of RDF and ShEx

We employed RDF and ShEx as machine-readable description frameworks to specify PoLyInfoRDF (the RDF version of PoLyInfo) and the ShEx schema, respectively. This section briefly outlines the frameworks.

2.1. RDF

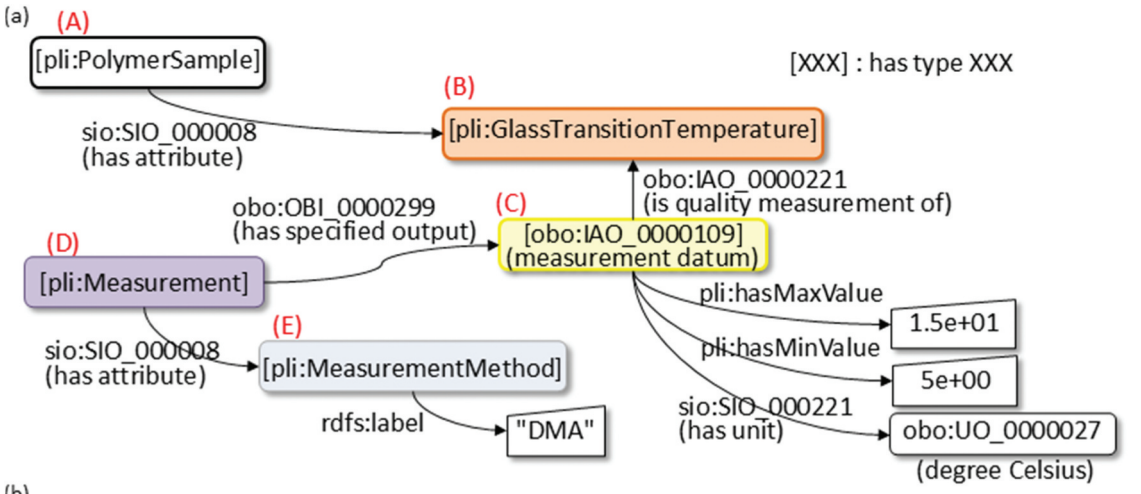
We created PoLyInfoRDF from PoLyInfo RDB according to the Resource Description Framework (RDF) based on the recommendations of the World Wide Web Consortium (W3C) [13]. RDF is a framework for representing information on the Web, where the core construct is an RDF graph, which is a set of triples, each consisting of a subject, predicate, and an object. An RDF graph can be depicted as a node and directed-arc diagram, in which each triple is represented as a node-arc-node link [4].

Figure 1a,b show a sample RDF graph expressing the glass transition temperature in PoLyInfo, represented in (a) graphical format and (b) text format (Turtle format). Note that the list of prefixes used in this paper, such as 'pli:', is presented in APPENDIX A, and the notation '[pli:PolymerSample]' (a class enclosed within square brackets) in an RDF node indicates that this node has the type pli:PolymerSample (same hereafter). Both (a) and (b) state that a polymer sample (denoted by node (A)) has a glass transition temperature property (B), the measured value of which is shown as node (C), where the actual values are between 5 and 15 degrees Celsius. These data are obtained through a measurement process (D), which employs a method called Dynamic Mechanical Analysis (DMA).

2.2. ShEx

ShEx is an RDF schema description language published by the World Wide Web Consortium (W3C) Shape Expressions Community Group [7]. 'It is not a W3C Standard nor is it on the W3C Standards Track'. [7] Although we could employ another analogous language, the Shapes Constraint Language (SHACL) [14], which was published as a W3C recommendation, we chose ShEx, considering its popularity (e.g. it is employed in Wikidata [15] and Gene Ontology [16]) and ease of use [17].

A ShEx schema is a collection of labeled shapes and node constraints. These can be used to describe and verify the nodes in RDF graphs. Figure 2 shows a sample of the ShEx schema that describes the structure of the RDF graph in Figure 1a,b. Here, (A)'–(F)' are labeled shapes that stipulate the conditions on their target nodes. For example (A)' is a shape labeled



(b)

```
(A)
sample:0076263-002-001-001 a pli:PolymerSample ;
    sio:SIO_000008 prop:0076263-002-001-001-0311-001_GlassTransitionTemperature .
(B)
prop:0076263-002-001-001-0311-001_GlassTransitionTemperature
    a pli:GlassTransitionTemperature .
(C)
prop:0076263-002-001-001-0311-001_GlassTransitionTemperatureMeasurementDatum
    a obo:IAO_0000109 ;
    pli:hasMaxValue 1.5e+01 ;
    pli:hasMinValue 5e+00 ;
    obo:IAO_0000221 prop:0076263-002-001-001-0311-001_GlassTransitionTemperature ;
    sio:SIO_000221 obo:UO_0000027 .
(D)
prop:0076263-002-001-001-0311-001_GlassTransitionTemperatureMeasurement
    a pli:Measurement ;
    obo:OBI_0000299
    prop:0076263-002-001-001-0311-001_GlassTransitionTemperatureMeasurementDatum ;
    sio:SIO_000008 [ a pli:MeasurementMethod ;
            rdfs:label "DMA" ] .(E)
```

Figure 1. Examples of (a) RDF graph and(b) RDF text (glass transition temperature).

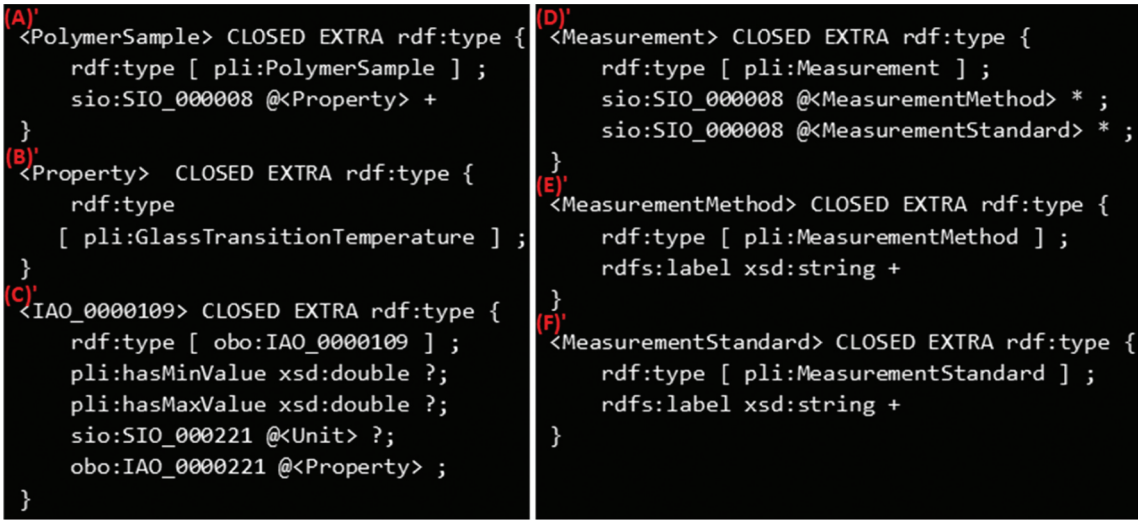


Figure 2. Example of ShEx schema.

<PolymerSample> and states that its target nodes (A) in Figure 1a,b should have the type pli:PolymerSample and should be linked through sio:SIO_000008(has attribute) to one or more nodes (indicated by the '+' sign) that are target nodes of the <Property> shape, where the '@' sign before the shape label indicates the reference to that shape. Similarly, <Property> shape (B)' states that its target nodes (B) should have the type pli:GlassTransitionTemperature. The shape <Unit> in (C') states a condition on unit nodes, whose code is omitted here for simplicity. Note that cardinality '?' and '*' appended to a link indicate that these links are 'optional' and 'allowed to appear any number (including 0) of times', respectively.

3. RDF representations of descriptors and properties

PoLyInfo provides a wide variety of data required for designing polymeric materials. These data were gathered from various aspects to cover the diverse characteristics of the polymers. In PoLyInfo, we describe these characteristics primarily using descriptors and properties. They form the core of PoLyInfo, and, in this section, we examine how they are represented as RDF graphs in PoLyInfoRDF.

3.1. Basic ideas

Generally, the various characteristics of an entity are described by a combination of attribute-value pairs [18]. For example, person A has the attributes of nationality, body height, and body weight, with values in Japan of 170 cm and 60 kg, respectively. Similarly, a polymer (sample) may have, among others, the attributes of a source-based name and melting temperature with values 'poly[ethene-co-(buta-1,3-diene)]' and 65 degrees Celsius. Here we refer to the attribute values as 'descriptors', implying that they carry information for describing the owner entity's characteristics from some particular aspects. In the above example, the two values are the descriptors of the source-based name and melting temperature attributes of the polymer (sample) in question.

Note the difference between the nature of source-based names and melting temperatures. We may say that the former are 'extrinsic' in that these names are given to polymers based on a kind of social agreement outside of their chemical or physical features. Meanwhile, the latter may be 'intrinsic' because they are inherent in polymers and directly mirror these features, explicitly describing the essential aspects of polymer behavior. In PoLyInfo, we roughly refer to the latter type of attributes of a polymer (sample) as 'properties' and they are of concern as targets of our consideration for elucidating the innate behaviors of polymers. For example, data gathered on the electrical conductivity of polymers play a significant role in examining their performance as conductive polymers. All the properties of PoLyInfo are listed in [19]. These are the attributes of a polymer (sample) that are the targets of our PoLyInfo data collection. For reference, they are a kind of 'intrinsic' attribute of a polymer and we selected them from among other attributes considering several factors other than the above 'innateness', including chemical or academic significance, industrial concerns, etc.

Definition: 'property' is an attribute listed in the PoLyInfo property list and 'descriptor' is a value of an attribute of some entity.

Note how these notions are embodied in our PoLyInfo ontology. Currently, each property in the PoLyInfo property list above in fact falls into one of the two categories, quality or disposition as referred to in the Basic Formal Ontology (BFO) [20], which is one

of our upper ontologies [11]. Thus we define the property class pli:Property in our ontology as a subclass of the union of quality (obo:BFO_0000019) and disposition (obo:BFO_0000016). Moreover, due to the fact that descriptor is a data value, particularly of an attribute, the pli:Descriptor class is, like measurement datum (obo:IAO_0000109), deployed under the data item (obo:IAO_0000027) class.

3.2. Basic RDF patterns for descriptors and properties

We designed RDF graphs for descriptors and properties based on diagrams (A-1), (A-2), and (B) in Figure 3, drawn with reference to the Chemical Information Ontology (CHEMINF) [21,22], Information Artifact Ontology (IAO) [23], and PubChemRDF [24–27], respectively. The basic RDF patterns of PoLyInfoRDF are shown in Figures 4 (C-1) and (C-2). As shown in these figures, (C-1) has the same structure as (A-1) and (A-2). However, (C-2) is similar to (B), and node (2), indicating the attributes of (C-1), is omitted. For properties, graph (C-1) applies and states that the polymer sample (1) has a property (e.g. melting temperature) (2), whose measurement datum (measured property value) is (3) measured with (4). For descriptors other than the property values, graph (C-2) applies and states that entity (1) owns descriptor (3) measured with (4). Because descriptor node (3) holds the value of a specific attribute (e.g. a source-based name) that can be inferred from the node type (pli:SourceBasedName), attribute node (2) may be omitted from the graph. Comparing the two graphs, we observe that while (C-2) is more concise and straightforward, (C-1) is rather complex, but more flexible, which is equipped with higher expressive capabilities because the property node (2) remains independent of a value node (3). Accordingly, for properties, we adopted (C-1) because, among other attributes, they would be of more interest to us and would be examined rather thoroughly. Hence, we should be ready to describe them from a wider range of aspects, using the more flexible pattern (C-1).

Note that in (C-1) obo:IAO_0000221 (is quality measurement of) has the range constraint that the object (melting temperature property node (2) here) should be a quality. Here, as mentioned in 3.1, each property in PoLyInfo is a kind of quality or disposition, causing a violation in case it is a disposition, which we would accept for the following reason. As for properties themselves we should care about the

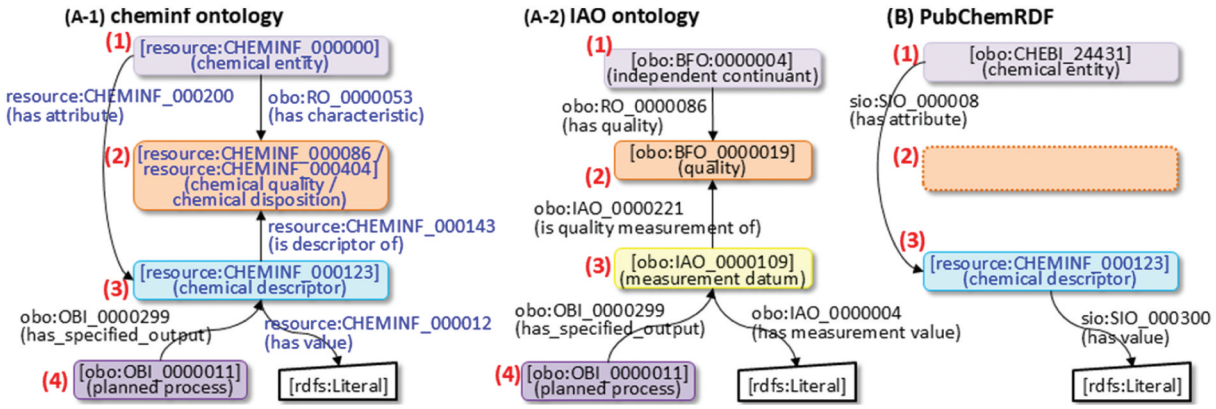


Figure 3. Basic RDF patterns for descriptors and properties (external ontologies).

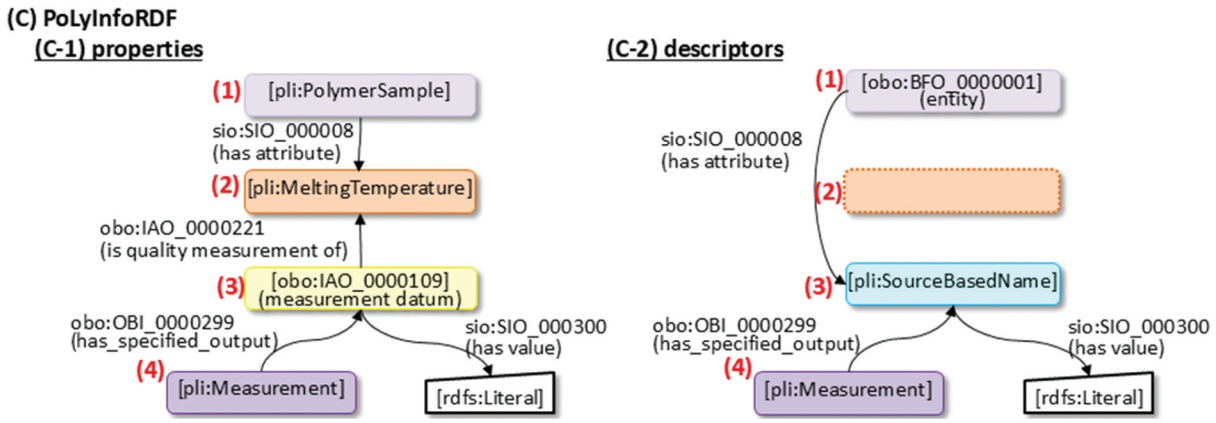


Figure 4. Basic RDF patterns for descriptors and properties (PoLyInfoRDF).

distinction between qualities and dispositions when examining their natures in various occasions. On the other hand, regarding their relationships with the measurement data (i.e. the 'is quality measurement of' link to the 'melting temperature' property node, in this case), this distinction seems to be of little significance, and for the sake of convenience we preferred treating them alike. For example, SPARQL queries for the PoLyInfoRDF often involve some particular properties and their measurement data (e.g. to find polymer samples whose melting temperatures have measurement dada over 100 degrees Celsius, or to find all the properties and their measurement data of a specific polymer sample). Such queries are likely to refer to the predicates (such as IAO_0000221) between properties and measurement dada. Then, although the above violation could be circumvented by using different predicates depending on whether the properties are actually a kind of quality or of disposition, it might complicate the queries and we think should be avoided.

Further note that, as in PoLyInfo(II) [11], we bring predicates from well-known external ontologies whenever possible to minimize the PoLyInfo specific ones in our ontology, paying attention to their domain and range constraints. For example, as mentioned above, we use IAO_0000221 under the above mentioned background. Moreover we employ sio:SIO_000008 (has attribute) in the Semanticscience Integrated Ontology (SIO) [28], for associating polymer samples with properties or descriptors as in (C-1)(C-2), as well as measurements with measurement methods as in Figure 1a where we regard these measurement methods as a kind of descriptor that describes the measurements in question. Also, taking into account that in the SIO source code SIO_000008 has no explicit range constraints, we accept it for use in these cases. Refer to Appendix B for more details of the RDF representations of descriptors and properties.

4. Descriptors and properties in PoLyInfoRDF schema

In this section, we reveal the content of the PoLyInfoRDF schema written in ShEx, focusing on how descriptors and properties are defined in the schema under the modular approach.

4.1. Features for modularization in ShEx

As mentioned earlier (1.2), modularization is a major design priority of the PoLyInfoRDF schema. Here, we introduce some of the relevant features of ShEx.

4.1.1. Recursion

In our approach, we often require a shape for RDF nodes, the type of which is a subclass of a particular root class. For example, polymerization nodes should have a type that is a subclass of pli:Polymerization (the root class), such as the pli:AdditionPolymerization and pli:RingOpeningPolymerization classes, as shown in Figure 5a (refer to the ontology in Figure 5b for the class hierarchy). The shape for these nodes would desirably be defined briefly as (1)' in Figure 5c, where, instead of exhaustively specifying every single subclass of pli:Polymerization, we simply employ the <PolymerizationSubclass> shape to collectively cover all of them. Here, we resort to the recursion feature in ShEx and define this shape recursively

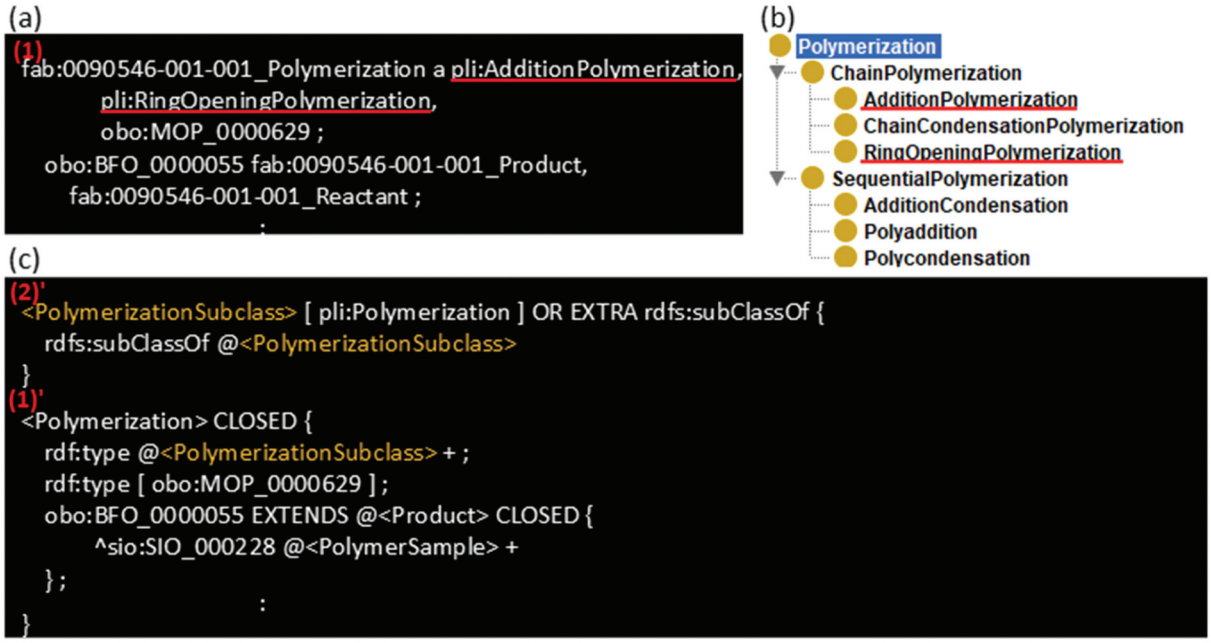


Figure 5. Recursion in ShEx (shape for subclasses). Examples in (a) RDF, (b) PoLyInfo ontology and (c) ShEx schema.

(i.e. the shape refers to itself in its definition), as in (2)' [29], leading to a clear and concise definition. Additionally, one might want to define these subclass shapes as something similar to rdfs:subClassOf*, a property path in SPARQL. However, because ShEx does not allow for such notations, special treatment, as described above, is required.

4.1.2. Importing schemas

In the modular approach, an entire schema is divided into several schemas, including those that are reusable. Under these circumstances, the IMPORT directive signifies the loading of other schemas, enabling the importing schema to reference the shapes defined in the imported schemas [30]. For example, Figure 6a shows a portion of Additive.shex, where two reusable schemas are imported and you are allowed to reference the shapes <Material> and <Additive> without definition, that are defined in those imported schemas such as materials.shex (Figure 6b).

4.1.3. Inheritance

ShEx contains constructs for inheritance (EXTENDS/RESTRICTS) that significantly facilitate the reuse of schemas [30]. Although they are described only in a draft report and not in the official specifications, they are so advantageous that we employed them in our schema design, leading to complete, concise, and straightforward schemas in a well-organized manner, which would facilitate understanding of the scientific data structure described by these schemas. We illustrate these constructs using a simple example. Suppose that Japanese and employees are both subordinate concepts of a person. A sample ShEx schema using inheritance is shown in Figure 7a, where each shape imposes the following conditions on the nodes in an RDF graph:

(1) Every Person node should have an attribute name and age, whose values are a string (possibly language-tagged) and an integer, respectively.
(2) Every Japanese node should meet the condition (1) for a Person node (stipulated by 'RESTRICTS @<Person>') and further the value of the name attribute should be a Japanese string. (further restriction on the triples in <Person>)
(3) Every Employee node should have the attributes in (1) for a Person node (stipulated by 'EXTENDS @<Person>') and further the employee_id attribute whose value is a string (extension of the triples in <Person>).

Here, (2) and (3) are equivalent to their inheritance-free counterparts, (2)' and (3)', in Figure 7b. Compared with (2)' and (3)', (2) and (3) explicitly state that the Japanese and Employee nodes inherit the characteristics of the Person nodes. Roughly speaking, RESTRICTS and EXTENDS each stipulate a condition as a Person node, which is further strengthened by additional requirements, making the Person node more specific. This is similar to

(a)

```
IMPORT <materials.shex>
IMPORT <mixtureRoles.shex>

START=@<PolymerSample>

<PolymerSample> CLOSED {
   rdf:type [ pli:PolymerSample ] ;
   obo:BFO_0000051 EXTENDS @<Material> CLOSED {
        sio:SIO_000228 @<Additive>
      } +
}
```

(b)

```
<Material> CLOSED EXTRA rdf:type {
   rdf:type [ pli:Material ] ;
   rdf:type [ obo:CHEBI_23367 ] ;

   rdfs:label xsd:string ? ;
   rdfs:comment xsd:string ?
}
```

Figure 6. Importing schemas in ShEx. Examples of (a) importing schema (Additive.shex) and (b) imported schema (materials.shex).

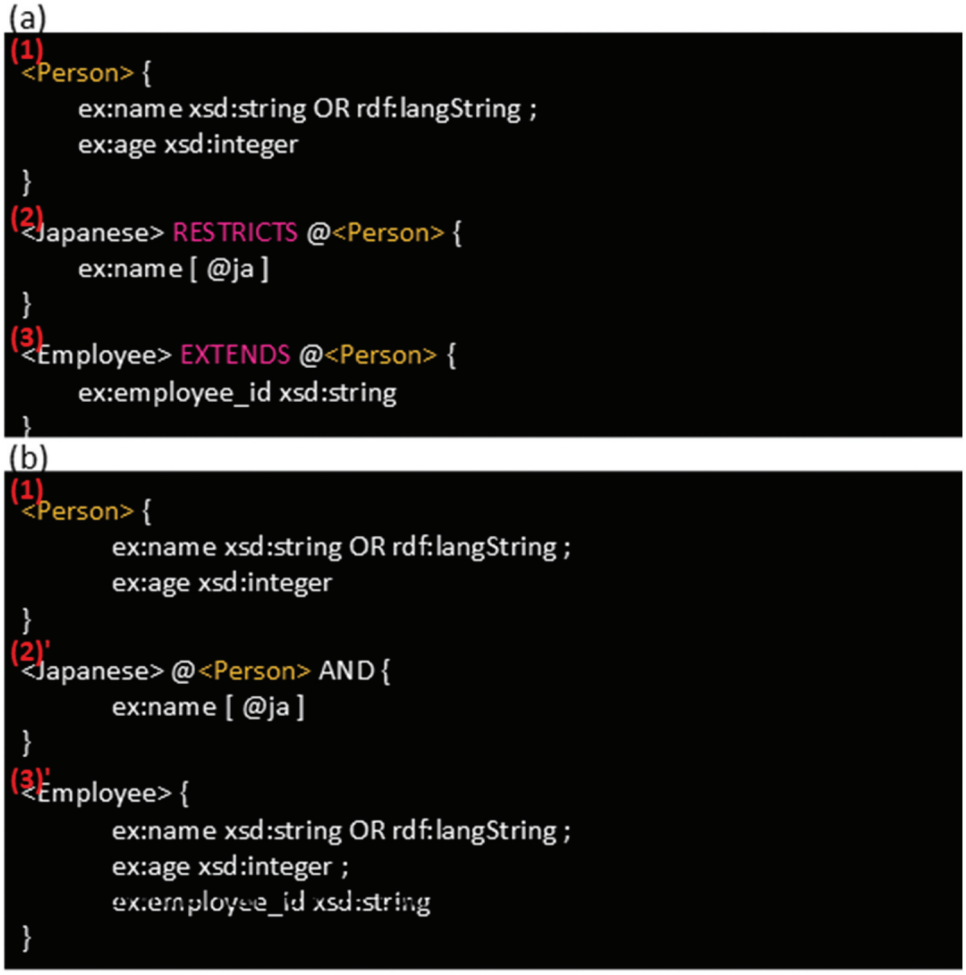


Figure 7. Inheritance in ShEx. Contrasting codes (a) with inheritance and (b) without inheritance.

inheritance in the object-oriented paradigm, and in our approach, it plays a significant role in modularization.

4.2. Details of the PoLyInfoRDF schema

In this section, we clarify the details of the PoLyInfoRDF schema, focusing on how modularization is realized by employing the above-mentioned ShEx features. In Appendix C the actual examples of reusing descriptor and property schemas defined under modularization are shown.

4.2.1. Directory structure

The entire set of PoLyInfoRDF schema files is divided into several directories. A brief description of each directory follows (for further details, see PoLyInfo(II) [11]).

(a) lib. This directory is a collection of ShEx files (termed 'component schemas') supposed to be imported and reused from other ShEx files. The shapes defined in these component schemas, 'component shapes', can be referenced throughout the entire PoLyInfo schema. The lib directory is divided into the following sub-directories:

- descriptors

Contains component schemas related to descriptors, which we described in 3.2, Appendix B1, e.g. the descriptor for a molecular structure such as 'molecular formula' (molecularDescriptors.shex).

- roles

Contains component schemas related to roles, e.g. the role in chemical reactions such as 'ambient gas' (chemicalReactionRoles.shex).

- others

Contains other component schemas (deployed directly under the lib directory), e.g. schemas related to values such as 'numeric value' (values.shex).

(b) masters. Contains schemas related to master information, which normalizes the chemical substances, constitutional units (CUs) of polymers, and

polymerization, e.g. master information on homopolymers (Homopolymer.shex).

(c) materials. Contains schemas related to material information, which includes additives, fillers, etc., e.g. material information on additives (Additive.shex).

(d) fabrications. Contains schemas related to fabrication information, which summarizes synthetic processes, e.g. fabrication information on polymerizations (PolymerizationInformation.shex).

(e) formations. Contains schemas related to formation information, which summarizes the higher-order structures of samples, e.g. formation information on crystallinity (Crystallinity.shex).

(f) properties. Contains schemas related to property information of the polymer samples, e.g. property information on the melting temperature (Melting Temperature.shex).

4.2.2. Descriptors

In this subsection we detail the schemas of descriptors.

(1) Basic formulation. As mentioned previously (3.1), in our study, each descriptor is a value of an attribute, and its RDF graph incorporates a value portion. Based on this observation, we define the shapes of the descriptors as follows: Figure 8 shows the hierarchy of and relationships among those shapes, where <ShapeValue> and <Descriptor> are the topmost shapes for values and descriptors. Under <Descriptor> the shapes for each descriptor pattern (refer to Figures B1 and B2 in Appendix B) are deployed. These shapes are abstract descriptor shapes (see below) and inherit, in addition to <Descriptor>, one of the value shapes used to define the value portion of the descriptor. For example, <StringDescriptor> inherits <StringValue>, which states that the string descriptor node in the RDF holds a string value. Any non-abstract descriptor shape in the PoLyInfo schema inherits one of these abstract descriptor shapes. For example, <SourceBasedName> and <Molecular Formula> inherit <StringDescriptor>, showing that they are defined by refining <StringDescriptor>.

Note here that we refer to the shapes that are used only as the upper of other shapes as 'abstract' ('abstract' in the sense of the object-oriented paradigm) and to the others as 'non-abstract'. We used these terms with reference to [30].

(2) Abstract descriptor shapes. Figure 9a shows the ShEx code for values. Each shape stipulates the value portions of the descriptors and property values (Figure B1–B4) and inherits the upper shape using the EXTENDS construct. For example, <StringValue> extends <ShapeValue> by adding a link carrying a string literal (possibly language-tagged). Figure 9b shows the abstract descriptor shapes. For example <StringDescriptor> inherits <StringValue> and <Descriptor>, stating that the string descriptor node in RDF holds a string value stipulated by <StringValue> and some descriptor type (in this Figure <Descriptor> imposes a type as stipulated by <DescriptorSubclass>, which is a subclass of pli: Descriptor, as in 4.1.1).

Figure 10a–e show sample ShEx codes for various kinds of non-abstract descriptor shapes. In Figure 10a, this shape is a kind of string descriptor shape, restricting the type in the upper <StringDescriptor> shape. In Figure 10b these shapes are a kind of numeric descriptor shape, restricting the type in <NumericDescriptor> . In Figure 10c, these shapes are a kind of discrete descriptor shape, restricting the type in <DiscreteDescriptor> . In

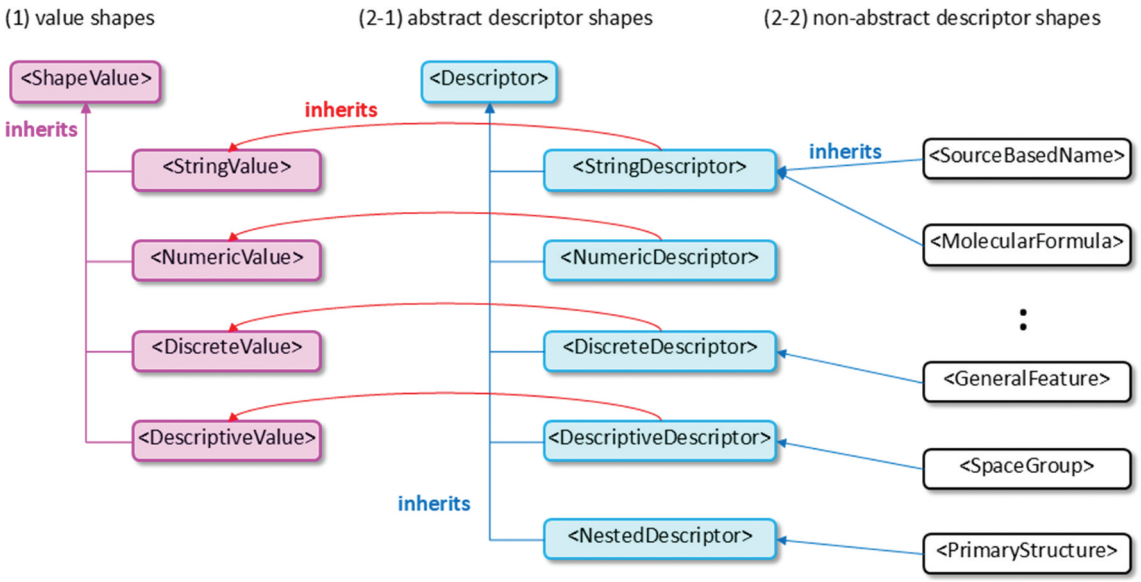


Figure 8. Hierarchies of value and descriptor shapes.

(a)

```
IMPORT <primitiveEntities.shex>

### values in general
<Value> @<NamedIndividual> OR @<ShapeValue>
<ShapeValue> {}
### string values ###
<StringValue> EXTENDS @<ShapeValue> CLOSED {
   sio:SIO_000300 xsd:string OR rdf:langString
}
### numeric values ###
<NumericValue> EXTENDS @<ShapeValue> CLOSED {
   (sio:SIO_000300 xsd:double OR xsd:integer ? |
   pli:hasMinValue xsd:double ?;
   pli:hasMaxValue xsd:double ?;
   pli:hasInequalitySign xsd:string ?) ;
   sio:SIO_000221 @<Unit> ?
}
### discrete values ###
<DiscreteValue> @<NamedIndividual>
   OR EXTENDS @<ShapeValue> {
      sio:SIO_000300 xsd:string OR rdf:langString ;
      owl:sameAs @<NamedIndividual> ?
   }
### description specified values ###
<DescriptiveValue> EXTENDS @<ShapeValue> CLOSED {
   sio:SIO_000255 @<Description>
}
### descriptions ###
<Description> CLOSED EXTRA rdf:type {
   rdf:type [ sio:SIO_000136 ] ;
   rdfs:comment xsd:string
}
### units ###
<UnitSubclass> [obo:UO_0000000 pli:Unit]
   OR EXTRA rdfs:subClassOf { rdfs:subClassOf @<UnitSubclass> }

<Unit> @<UnitSubclass>
   OR CLOSED EXTRA rdf:type { rdf:type [ obo:UO_0000000 ];
        oboInOwl:hasExactSynonym xsd:string }
```

(b)

```
IMPORT <values.shex>

### subclasses of pli:Descriptor
<DescriptorSubclass> [ pli:Descriptor ]
   OR EXTRA rdfs:subClassOf { rdfs:subClassOf @<DescriptorSubclass> }

### descriptors in general
<Descriptor> EXTRA rdf:type {
   rdf:type @<DescriptorSubclass>     # type : subclass of pli:Descriptor
}

<NumericDescriptor> EXTENDS @<NumericValue> EXTENDS @<Descriptor> {}
<StringDescriptor> EXTENDS @<StringValue> EXTENDS @<Descriptor> {}
<DiscreteDescriptor> EXTENDS @<DiscreteValue> EXTENDS @<Descriptor> {}
<DescriptiveDescriptor> EXTENDS @<DescriptiveValue> EXTENDS @<Descriptor> {}
<NestedDescriptor> EXTENDS @<Descriptor> { obo:BFO_0000051 @<Descriptor> * }
```

Figure 9. Shapes for (a) values and (b) descriptors.

Figure 10d this shape is a kind of descriptive descriptor shape, restricting the type in <DescriptiveDescriptor>. Figure 10e shows the shapes related to primary structure, that are a kind of nested descriptor shape. They inherited <NestedDescriptor> using RESTRICTS or EXTENDS. Some impose both restrictions and extensions on the upper shapes. For example, <ChainStructure> inherits <NestedDescriptor> with a condition that the RDF nodes in question should have pli:ChainStructure type and an additional link holding a string literal. The RDF graph of the nodes related to primary structure is shown in Figure B2. In fact, the RDF file for this graph is divided into several for the sake of human readability (rather than for modularization or reusability); accordingly, the schema is also defined separately, including that shown in Figure 10e.

(3) Distinction of hierarchies of classes and shapes. Although the hierarchy of ontology classes and that of shapes seem somewhat analogous, they are built from

(a)

```
IMPORT <descriptors.shex>

<SourceBasedName> RESTRICTS @<StringDescriptor> EXTRA rdf:type {
  rdf:type [ pli:SourceBasedName ]
}
```

(b)

```
IMPORT <descriptors.shex>

<Frequency> RESTRICTS @<NumericDescriptor> EXTRA rdf:type {
  rdf:type [ pli:Frequency ]
}

<Wavelength> RESTRICTS @<NumericDescriptor> EXTRA rdf:type {
  rdf:type [ pli:Wavelength ]
}
```

(c)

```
IMPORT <descriptors.shex>

<SampleType> RESTRICTS @<DiscreteDescriptor> EXTRA rdf:type {
  rdf:type [ pli:SampleType ] ;
  owl:sameAs EXTRA rdf:type { rdf:type [ pli:SampleType ] } ?
}

<UlFlammabilityCodeRatingType> RESTRICTS @<DiscreteDescriptor> EXTRA rdf:type {
  rdf:type [ pli:UlFlammabilityCodeRatingType ] ;
  owl:sameAs EXTRA rdf:type { rdf:type [ pli:UlFlammabilityCodeRatingType ] } ?
}
```

(d)

```
IMPORT <descriptors.shex>

<SpaceGroup> RESTRICTS @<DescriptiveDescriptor> EXTRA rdf:type {
  rdf:type [ pli:SpaceGroup ]
}
```

(e)

```
IMPORT <descriptors.shex>

<PrimaryStructure> RESTRICTS @<NestedDescriptor> EXTRA rdf:type {
  rdf:type [ pli:PrimaryStructure ]
}

<BranchingAndCrosslinking> RESTRICTS @<NestedDescriptor> EXTRA rdf:type {
  rdf:type [ pli:BranchingAndCrosslinking ]
}

<ChainStructure> RESTRICTS @<NestedDescriptor> EXTRA rdf:type {
  rdf:type [ pli:ChainStructure ]
} AND EXTENDS @<NestedDescriptor> CLOSED {
  sio:SIO_000300 xsd:string
}

<EndGroupStructure> RESTRICTS @<NestedDescriptor> EXTRA rdf:type {
  rdf:type [ pli:EndGroupStructure ]
} AND EXTENDS @<NestedDescriptor> CLOSED {
  rdfs:comment xsd:string ?
}

<EndGroupConnection> RESTRICTS @<NestedDescriptor> EXTRA rdf:type {
  rdf:type [ pli:EndGroupConnection ]
}
```

Figure 10. ShEx codes of non-abstract descriptor shapes. Examples of (a) string descriptors, (b) numeric descriptors, (c) discrete descriptors, (d) descriptive descriptors and (e) nested descriptors.

completely different perspectives with distinct motives. That is, whereas the former is based on concepts denoted by classes ('concept hierarchy'), the latter is based on RDF node patterns specified by shapes for representing concepts (representation-based hierarchy or 'schema hierarchy'). For example, in the PoLyInfo ontology, the pli:SourceBasedName class is a subclass of the pli:NameDescriptor class because the source-based name is a kind of name descriptor (a decision based on the concepts denoted by these classes), for the sake of clarifying the relationships among concepts. On the other hand, the <SourceBasedName> shape inherits (equivalently, is subordinate to) the <StringDescriptor> shape because the RDF node pattern specified by the former, which includes a string value, is a refinement of that by the latter, which makes the latter reusable by restricting or extending the triples in it.

The distinction between these hierarchies implies that specifying a concept (positioning it in the conceptual hierarchy) and representing it (defining its shape for the RDF nodes) are separate matters. In fact, the <StringDescriptor> shape, inherited by the shape of the source-based name descriptor as above, is also inherited by the shape of, for example, the molecular formula descriptor, which belongs to conceptually another descriptor category (molecular descriptor, not name descriptor). Thus, identifying and standardizing the common portions among the RDF nodes, regardless of the conceptual hierarchy, leads to the extraction of relevant component shapes that are reusable throughout the schemas.

4.2.3. Properties

In PoLyInfoRDF, three kinds of nodes appear in combination to represent the properties: (1) property, (2) measurement datum, and (3) measurement (cf. Appendix B, Figures B3,B4). Similar to descriptors, we largely resort to inheritance to define the schemas of properties.

Figure 11 illustrates the basic formulation of the shapes with respect to properties. The formulation is vertically partitioned into three groups, each containing shapes for the nodes of (1) property, (2) measurement datum, and (3) measurement. Furthermore, it has a three-layer structure, and in principle, each shape in the lower layer inherits that in the upper layer, as indicated by the thin arrows. Layer1 is a core-level layer, where each shape defines core information, such as type, that stipulates a condition on nodes (1), (2), and (3), such that their types are a subclass of pli:Property, obo:IAO_0000109, and pli:Measurement, respectively. (The type of a node is enclosed within square brackets). Layer2 is a base-level layer, where each shape inherits its upper and defines the links among nodes (1), (2), and (3) with obo:IAO_0000221 (is quality measurement of) and obo:OBI_0000299 (has specified output), as shown by the thick arrows. Moreover, the abstract shapes for a variety of property values in Appendix B2 are defined by inheriting <IAO_0000109_base>. For example, <IAO_0000109_base_numeric> is the abstract shape for numeric measurement data, which inherits <IAO_0000109_base> with incorporation of a numeric value. The shapes in this layer were further inherited from those in layer3. The ShEx codes for the shapes in layer1 and 2 are shown in Figure 12.

In layer3, non-abstract shapes are defined for each individual property, basically inheriting shapes from layer2 and imposing further conditions of their own. Particularly, the measurement datum shapes (<IAO_0000109>) in layer3 inherit one of three upper ones depending on the nature of their values. For example, the shape for the melting temperature property inherits <Property_base> shape and confines the type of the property nodes in question to pli:MeltingTemperature. Moreover, this property has numeric values; thus, its measurement datum shape inherits <IAO_0000109_base_numeric>, the numeric version of <IAO_0000109_base>. Its measurement

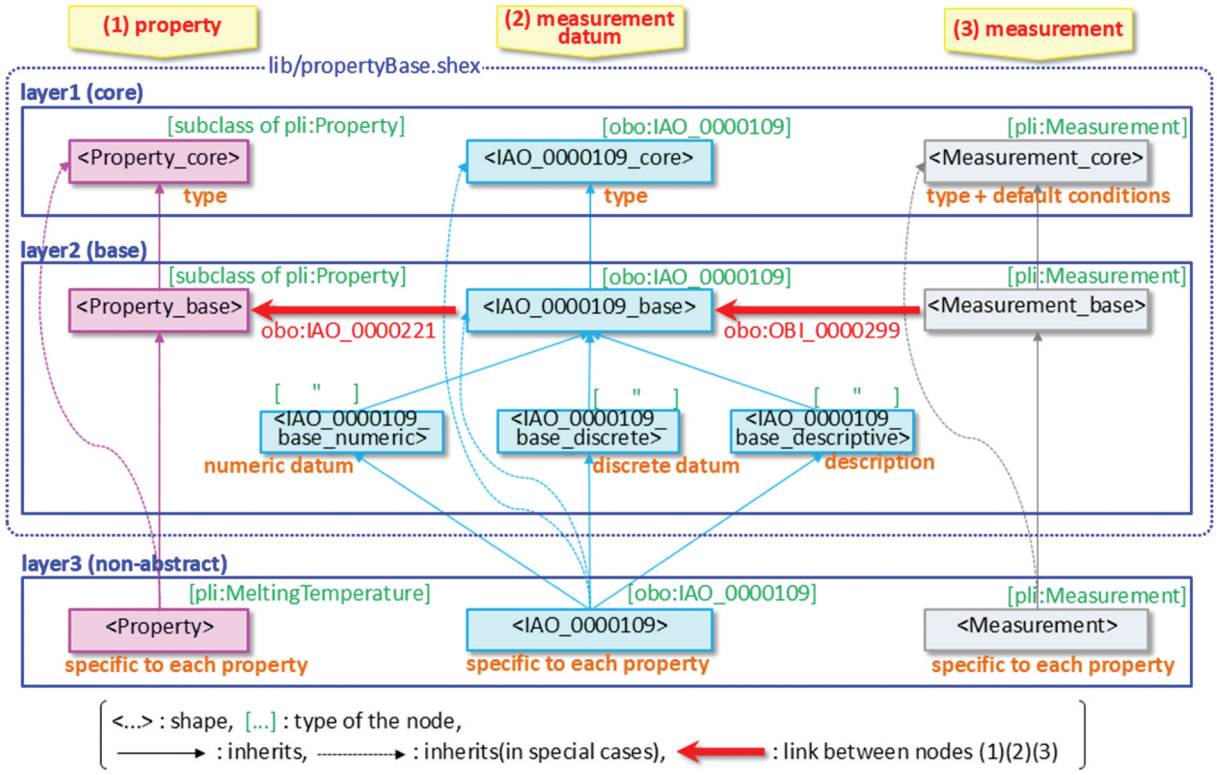


Figure 11. Basic formulation of property shapes.

```
IMPORT <values.shex>
IMPORT <Descriptors.shex>
IMPORT <physicalDescriptors.shex>

<PropertySubclass> [ pli:Property ]
   OR EXTRA rdfs:subClassOf { rdfs:subClassOf @<PropertySubclass> }

<Property_core> EXTRA rdf:type {
   rdf:type @<PropertySubclass>
}

<Property_base> EXTENDS @<Property_core> {
   ^obo:IAO_0000221 @<IAO_0000109_base>
}

<IAO_0000109_core> EXTRA rdf:type {
   rdf:type [ obo:IAO_0000109 ]
}

<IAO_0000109_base> EXTENDS @<IAO_0000109_core> {
   obo:IAO_0000221 @<Property_base> ;
   ^obo:OBI_0000299 @<Measurement_base> ?
}

<IAO_0000109_base_numeric> EXTENDS @<NumericValue>
   EXTENDS @<IAO_0000109_base> CLOSED {}

<IAO_0000109_base_discrete> EXTENDS @<IAO_0000109_base> {
   sio:SIO_000008 @<DiscreteDescriptor>
}

<IAO_0000109_base_descriptive> EXTENDS @<DescriptiveValue>
   EXTENDS @<IAO_0000109_base> CLOSED {}

<Measurement_core> CLOSED EXTRA rdf:type {
   rdf:type [ pli:Measurement ] ;
   sio:SIO_000008 @<MeasurementMethod> * ;
   sio:SIO_000008 @<MeasurementStandard> * ;
   sio:SIO_000008 @<Description> * ;
   sio:SIO_000008 (@<Temperature> OR @<RelativeHumidity> OR
@<Frequency> OR @<Voltage> OR @<Duration>) *
}

<Measurement_base> EXTENDS @<Measurement_core> CLOSED {
   obo:OBI_0000299 @<IAO_0000109_base> +
}

<MeasurementMethod> CLOSED EXTRA rdf:type {
   rdf:type [ pli:MeasurementMethod ] ;
   rdfs:label xsd:string +
}

<MeasurementStandard> CLOSED EXTRA rdf:type {
   rdf:type [ pli:MeasurementStandard ] ;
   rdfs:label xsd:string +
}
```

Figure 12. Property shapes in layer1 and 2.

shape inherits <Measurement_base> without any additional condition.

Regarding these RDF representations of properties let us add some supplemental comments:

- As a class of property values, we employ the class obo:IAO_0000109 in the Information Artifact Ontology (IAO), reflecting our design policy to comply with global standard ontologies as much as possible without introducing PoLyInfo-specific items such as the pli:PropertyValue class.
- The shapes in layer1 and 2 are 'abstract' in the sense noted in 4.2.2(1) and defined under the lib directory intended to be reusable as component shapes (4.2.1(a)). The shapes in layer3 are non-abstract and defined under other directories, such as fabrications or properties, etc., as 'users' of these component shapes.

- As shown in layer3, although the property nodes have their own types, such as pli: MeltingTemperature, which depend on each property ('strongly' typed), the measurement datum and measurement nodes have the types obo:IAO_0000109 and pli:Measurement in common among the properties ('weakly' typed). This is because the property nodes denote the properties of polymers, which may take on various complicated characteristics and require elaborate treatment depending on the type of property. Hence, we applied 'fine-grained' typing to the property nodes. However, the property value and measurement nodes behave simply as containers of values and measurement conditions, and we observed that for these nodes, uniform treatment, regardless of the property type, may be sufficient without further strong typing. Even when necessary, this can be deduced from the type of associated property nodes.

5. Discussions

We explained how to define ShEx schemas for descriptors and properties in the context of modularization. In this section, we assess these efforts from the following perspectives: (1) the effects of modularization, (2) applications of the schemas, (3) remaining issues to be solved, and (4) the future prospects of our work.

5.1. Effects of modularization

One of the major purposes of modularization is to extract common portions as reusable components in a generic form to improve their reusability. In our ShEx schemas, the shapes defined under the lib directory are reusable components (component shapes, 4.2.1 (a)), which are referenced from other shapes. To assess the reusability of these component shapes, three types of shape references were considered: through IMPORT (type I), RESTRICTS (type II), and EXTENDS (type III). Here, we distinguish reusability arising from file division (type I) and inheritance (types II-III). In Figure 13, the lines marked (1), (2), and (3) are type I references to the shape <DiscreteDescriptor> which is defined in the imported descriptors.shex. The lines marked (2) and (3) are both type I and type II-III references to that shape; that is, this distinction is not exclusive.

Among these references, those of type I may be essential for reusability because they allow the sharing of shapes across multiple ShEx files, and we assessed their effect from the aspect of compacting the entire schema size. To estimate this effect, we used the following figures:

- (A) 560 lines (excluding blanks and comments)

The actual total number of lines in the ShEx schema files under the lib directory, which define the component shapes.

- (B) 2,927 lines

The actual total number of lines in ShEx schema files under directories other than lib.

- (C) 1470 lines

The estimated increase in the total number of lines to eliminate the use of type I references; that is, to replace type I references in ShEx schema files (other than the component schemas) with their equivalent code fragments (i.e. the bodies of their referenced shapes).

- (D) 4,397 lines (=(B)+(C))

Estimated total number of lines for all ShEx schema files without using type I references.

- (E) 0.21 (= 1-((A)+(B))/(D) = 1–3,487/4,397)

The figure (E) shows that the use of type I references condensed the schema codes by 21%. Although the main purpose of this work is to define a schema in

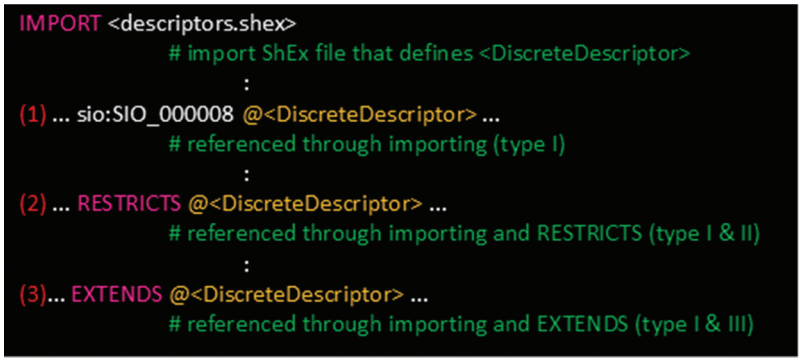


Figure 13. Types of shape references (type I, II, III).

a well-organized manner under a modular approach, condensing the schema size to a certain degree may lead to benefits such as enhancing the quality and reducing costs, as is the case with modularization as a whole in software engineering. As for the type II-III references they have the effect of explicitly stating the hierarchy of shapes through the two kinds of inheritance, RESTRICTS and EXTENDS. Although this seems difficult to evaluate quantitatively, we can surely say that they contribute remarkably to the readability and clarity of the entire schema (refer to the example in 4.1.3).

5.2. Applications of the schema

Originally, we defined the PoLyInfoRDF schema primarily for validation (verifying the validity of the PoLyInfoRDF data against the ShEx schema). Now, we are considering other applications, such as applying the schema to assist users in writing SPARQL queries. In Figure 14, for the input user query (a), the SPARQL query (b) for PoLyInfoRDF store is generated (refer to Figure 1 for the target graph structure), where symbols starting with '_', such as _smpl, denote variables.

In (a), ***path***(X, Y) is a command used to generate a SPARQL string that determines a path from node X to Y in the target RDF graph, like the underlined portions in (b). Here, PolymerSample (_smpl) and IAO_0000109(_datum, pli:GlassTransitionTemperature) indicate the node _smpl of the polymer sample and the node _datum of IAO_0000109(measurement datum), which is the value of the glass transition temperature. Note that temporary variable(s) (?v_2867) for the node(s) between _smpl and _datum in the path is automatically introduced in (b). Using this command, the path-(s) in question (Figure 1a (A)–(B)–(C)) are found without the user's awareness of the details of the RDF, such as the in-between node(s) ((B)).

We believe that one of the most monotonous burdens in writing SPARQL queries is specifying a path between two nodes in the RDF graph, especially in cases involving lengthy and complicated paths arising from, for example, data linkages with other external RDF stores. Hence, implementing these ***path*** commands would be beneficial for SPARQL users. Under these considerations, to translate ***path*** commands into equivalent SPARQL strings, we are investigating the application of ShEx schemas that formally specify the RDF node configurations in a graph required to identify the path in question.

Figure 15 outlines the current trial implementation. In this figure, (1) and (2) show the user query and generated SPARQL, respectively, as described above. The SPARQL string (2) is automatically converted from the user query (1) (arrow <1>) under the control of the Prolog [31,32] code (4) (<2>) and applied to PoLyInfoRDF stores (<3>). Here, (4) is a Prolog code containing Definite Clause Grammar (DCG) [33,34] rules, and is created based on the ShEx schema (3). For the outline of our current trial implementation, illustrative code fragments of (3) and (4) are shown in Appendix D.

(a)

```
select(distinct, _smpl, _value), where(
        path(PolymerSample(_smpl),
                IAO_0000109(_datum,
                        pli:GlassTransitionTemperature)),
        has_max_value(_datum, _value),
        filter(_value, '>', 100)
), [limit,10].
```

(b)

```
select distinct ?smpl ?value where {
        ?smpl rdf:type pli:PolymerSample.
        ?smpl sio:SIO_000008 ?v_2876.
        ?v_2876 rdf:type pli:GlassTransitionTemperature.
        ?v_2876 ^obo:IAO_0000221 ?datum.
        ?datum rdf:type obo:IAO_0000109.
        ?datum pli:hasMaxValue ?value.
        filter (?value > 100)
} limit 10
```

Figure 14. SPARQL generation from user queries. Examples of (a) input user query and (b) generated SPARQL (formatted by hand for readability).

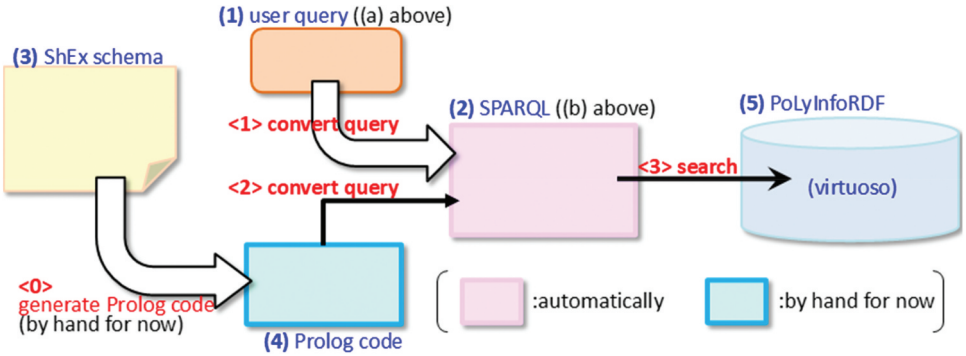


Figure 15. Trial implementation of SPARQL generation.

Thus far, the Prolog code (4) was created from the ShEx schema (3) thoroughly by hand; however, as in Appendix D, there seems to be a certain regularity in this conversion; hence, semi-automatic or fully automatic conversion may be possible by referencing schemas to support SPARQL writing. Additionally, in the future, by writing queries using generative Artificial Intelligence (AI) from a relevant set of schemas, a course of heuristic interactions with AI is expected to enable us to obtain the required query without exact knowledge of the details of the target RDF graph.

5.3. Remaining issues to be solved

We have devoted considerable effort to building the PoLyInfoRDF schema; however, some issues remain unresolved. In this section, we discuss one of them, the issue of shape name collisions. Figure 16 shows the ShEx code fragments for the (a) melting temperature and (b) glass transition temperature properties, where shapes with identical names carrying different contents can be seen. For example, the two <Property> shapes have a common name but hold their own distinct types: pli:MeltingTemperature and pli:GlassTransitionTemperature, causing a kind of name collision. Also regarding the two <PolymerSample> shapes, they literally have the same contents, whereas in reality they do not because of the above difference in their respective <Property> shapes referenced from inside.

At present, the issue of these shape name collisions is not as much of a problem because in our current schema design, the scope of shape names is local and limited to a single ShEx file, in which the uniqueness of shape names is preserved (locally unique). Hence, as long as the schema files are processed individually, as when validating RDF graphs in sequence against their respective ShEx schema files, name collisions do not occur. Strictly speaking, the above local uniqueness applies only to shapes other than the component shapes. That is, each schema file is processed along with its imported files (component schemas), which define the component shapes. Thus, for component shapes, we assume that their names have a global scope, ensuring that they are unique throughout the PoLyInfoRDF schema (globally unique).

Considering the various applications, we suspect that there might be cases in which more than one schema file is considered at a time, such as when applying ShEx schemas to generate Prolog codes for

(a)

```
IMPORT <propertyBase.shex>

<PolymerSample> CLOSED {
   rdf:type [ pli:PolymerSample ] ;
   sio:SIO_000008 @<Property> +
}

<Property> RESTRICTS @<Property_base> {
   rdf:type [ pli:MeltingTemperature ] ;
}

<IAO_0000109> @<IAO_0000109_base_numeric>

<Measurement> @<Measurement_base>
```

(b)

```
IMPORT <propertyBase.shex>

<PolymerSample> CLOSED {
   rdf:type [ pli:PolymerSample ] ;
   sio:SIO_000008 @<Property> +
}

<Property> RESTRICTS @<Property_base> {
   rdf:type [ pli:GlassTransitionTemperature ] ;
}

<IAO_0000109> @<IAO_0000109_base_numeric>

<Measurement> @<Measurement_base>
```

Figure 16. Issue of shape name collisions. ShEx codes for (a) melting temperature and (b) glass transition temperature.

query conversions (5.2, Figure 14). Referring to the example in Figure 16, imagine a query involving both properties, such as searching for polymer samples with the property values 60 and 80 degrees Celsius each. To recognize the RDF structures of both properties, their respective ShEx files need to be examined together, causing name collisions for shapes such as <Property>.

Therefore, to generalize the use of ShEx, we need measures to distinguish shapes with the same name. One possibility would be to rename these shapes (e.g. <MeltingTemperatureProperty> and <GlassTransitionTemperatureProperty>) or place them into distinct name spaces, which we consider to be unfavorable because it would overly complicate the schemas and affect their readability or clarity. To process multiple ShEx files simultaneously, we propose the following ShEx enhancements.

- Controlling the scope of each shape name, for example, enabling users to specify whether it is global or local, to instruct the ShEx processor to regard shapes with the same name defined in different ShEx files as separate shapes.
- Shapes may have parameters for arbitrary purposes, particularly for distinguishing shapes with the same name. For example assume that the property shape is defined as being equipped with parameters, such as <Property>(t), which is supposed to serve as a kind of shape template where t is a type parameter, which would result in its two instances <Property>(pli:MeltingTemperature) and <Property>(pli:GlassTransitionTemperature) handled as separate shapes.

5.4. Future prospects

In this section we express our views on the future potential of our modular approach for defining RDF schemas of your own in general. Although the original motivation for our schema modularization was to define the PoLyInfoRDF schema, the basic design strategy is widely applicable to other studies. In particular, we classified diverse forms of values into several categories, such as numeric, string, discrete, descriptive, etc., and defined their shapes (schemas) for each category, which, in a sense, standardized their RDF representations. Subsequently, inheriting (reusing) these shapes based on this classification drastically simplified the definition, in a well-organized manner, of other shapes carrying these values, such as those for descriptors and measurement data (property values).

Here, the above categories seem to be basically common among values in many cases and so would their shapes, where specifically introducing the 'discrete' category allows for a systematic approach for handling values under a uniform framework, including enumerations for grouping or grading, such as values of color (red, blue,. . .), in the sense noted in Appendix B1(3). Conventionally, these enumeration values are commonly encountered in the field, but they seemingly take on different natures and tend to be thought of as requiring treatment totally distinct from that of ordinary numeric or string values. In this respect, our approach would be well adopted in a wide range of fields, which involve various types of scientific data represented in the RDF format, and we expect further development and generalization of our research toward the future would contribute to the construction of generic libraries available for defining ShEx schemas in general. These libraries would store reusable component shapes, particularly for data values such as those mentioned above, which would contribute to the standardization of RDF representation of various types of scientific data in the relevant fields.

6. Conclusions

Motivated by recent progress in data-driven technologies, we developed PoLyInfoRDF and its ShEx schema, with the aim of making the PoLyInfo data machine-readable and further machine-understandable. Among these efforts, this study focused on the way in which the schemas are defined for descriptors and properties, both of which are the core of PoLyInfo data and are crucial for describing the various chemical or physical behaviors of polymers. We designed these schemas by following a 'modular approach', which defines component schemas as a library that is reusable in all the schemas. During the design, we took full advantage of the features of ShEx for modularization, such as inheritance, thereby resulting in complete, concise, and straightforward schemas. This success highlights functions that should be included in the official ShEx specifications in the future. We then assessed the results of building these schemas, including the effects of our modular approach and some future challenges, such as issues that may arise when applying ShEx schemas to assist SPARQL users and shape-name collisions.

Although many issues remain to be solved, including the above-mentioned ones, we have progressed significantly towards fulfilling the original goal of making the PoLyInfo data machine-readable and further machine-understandable. We expect these efforts to encourage various external RDF databases, whether public or proprietary, to establish data linkages with PoLyInfoRDF to fully utilize the rich collection of polymer data in PoLyInfo, which is foreseen to contribute considerably to the field of materials exploration. Furthermore, our modular approach is applicable

to a wide range of fields involving the RDF representation of scientific data. Developing our approach in the future could contribute to the standardization of scientific data representation in RDF by providing reusable schemas as a library that would subsequently enable users to define their own schemas in general. We consider our work to be a promising starting point for this challenging goal.

Acknowledgements

This study was partially supported by

- MEXT Program: Data Creation and Utilization-Type Material Research and Development Project Grant Number JPMXP1122714694
- Council for Science, Technology and Innovation (CSTI), Cross-ministerial Strategic Innovation Promotion Program (SIP), the 3rd period of SIP 'Materials Informatics Infrastructure Linkage and Human Resource Development for Fostering Material Unicorns' (Funding agency: NIMS)

This work was conducted using Protégé [35] and its outputs are used in some of the figures in this paper.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

The work was supported by the MEXT Program: Data Creation and Utilization-Type Material Research and Development Project [JPMXP1122714694]; Council for Science, Technology and Innovation (CSTI), Cross-ministerial Strategic Innovation Promotion Program (SIP), the 3rd period of SIP 'Materials Informatics Infrastructure Linkage and Human Resource Development for Fostering Material Unicorns' (Funding agency: NIMS) JPJ012308.

ORCID

Masashi Ishii http://orcid.org/0000-0003-0357-2832

Data availability statement

The data that support the findings of this study are openly available in the Materials Data Repository of NIMS at PoLyInfo Knowledge Collection, https://doi.org/10.48505/nims.4413 The current versions are https://doi.org/10.48505/nims.5396, reference number [6] and https://doi.org/10.48505/nims.5395, reference number [8].

References

[1] PoLyInfo [Internet]. Tsukuba, Japan: PoLyInfo; [cited 2025 Mar 19]. Available from: https://polymer.nims.go.jp/

[2] Ishii M, Ito T, Sado H, et al. NIMS polymer database PoLyInfo (I): an overarching view of half a million data points. Sci Technol Adv Mater Methods. 2024;4(1). doi: 10.1080/27660400.2024.2354649

[3] D2RQ [Internet]. San Francisco (CA): GitHub, Inc.; [cited 2025 Mar 19]. Available from: http://d2rq.org/

[4] RDF 1.1 Concepts and Abstract Syntax [Internet]. Wakefield (MA): World Wide Web Consortium; [cited 2025 Mar 19]. Available from: https://www.w3.org/TR/rdf11-concepts/

[5] RDF 1.2 Schema [Internet]. Wakefield (MA): World Wide Web Consortium; [cited 2025 Mar 19]. Available from: https://www.w3.org/TR/rdf12-schema/

[6] PoLyInfo Conceptual Schema Version 1.1. Tsukuba, Japan: PoLyInfo; [cited 2025 Jun 27]. doi: 10.48505/nims.5396 [Internet].

[7] Shape Expressions Language 2.1, Final Community Group Report 8 October 2019 [Internet]. Wakefield (MA): World Wide Web Consortium; [cited 2025 Mar 19]. Available from: http://shex.io/shex-semantics/

[8] PoLyInfo Ontology Version 1.2. Tsukuba, Japan: PoLyInfo; [cited 2025 Jun 27]. doi: 10.48505/nims.5395 [Internet].

[9] Web Ontology Language (OWL) [Internet]. Wakefield (MA): World Wide Web Consortium; [cited 2025 Mar 19]. Available from: https://www.w3.org/OWL/

[10] OWL 2 Web Ontology Language, Structural Specification and Functional-Style Syntax (Second Edition) [Internet]. Wakefield (MA): World Wide Web Consortium; [cited 2025 Mar 19]. Available from: https://www.w3.org/TR/owl2-syntax/

[11] Ishii M, Ito T, Sakamoto K. NIMS polymer database PoLyInfo (II): machine-readable standardization of polymer knowledge expression. Sci Technol Adv Mater Methods. 2024;4(1). doi: 10.1080/27660400.2024.2354651

[12] SPARQL 1.1 Query Language [Internet]. Wakefield (MA): World Wide Web Consortium; [cited 2025 Mar 19]. Available from: https://www.w3.org/TR/sparql11-query/

[13] The World Wide Web Consortium [Internet]. Wakefield (MA): World Wide Web Consortium; [cited 2025 Mar 19]. Available from: https://www.w3.org/

[14] Shapes Constraint Language (SHACL) W3C Recommendation 20 July 2017 [Internet]. Wakefield (MA): World Wide Web Consortium; [cited 2025 Mar 19]. Available from: https://www.w3.org/TR/shacl/

[15] Wikidata:Database reports/EntitySchema directory [Internet]. San Francisco (CA): Wikimedia Foundation, Inc.; [cited 2025 Mar 19]. Available from: https://www.wikidata.org/wiki/Wikidata:Database_reports/EntitySchema_directory

[16] GO_Shapes (Gene Ontology) [Internet]. San Francisco (CA): GitHub, Inc.; [cited 2025 Mar 19]. Available from: https://github.com/geneontology/go-shapes

[17] ShEx & SHACL compared [Internet]. London(UK): Figshare.; [cited 2025 Mar 19]. Available from: https://figshare.com/articles/presentation/ShEx_and_SHACL_compared/13174583

[18] Name–value pair [Internet]. San Francisco (CA): Wikipedia; [cited 2025 Mar 19]. Available from: https://en.wikipedia.org/wiki/Name%E2%80%93value_pair

[19] PoLyInfo Property List [Internet]. Tsukuba, Japan: PoLyInfo; [cited 2025 Mar 19]. Available from: https://polymer.nims.go.jp/PoLyInfo/guide/en/term_polymer.html#chap21
[20] Basic Formal Ontology 2.0, SPECIFICATION AND USER'S GUIDE [Internet]. San Francisco (CA): GitHub, Inc.; [cited 2025 Jun 26]. Available from: https://github.com/BFO-ontology/BFO/blob/master/docs/bfo2-reference/BFO2-Reference.pdf
[21] Hastings J, Chepelev L, Willighagen E, et al. The Chemical Information Ontology: Provenance and Disambiguation for Chemical Data on the Biological Semantic Web. PLoS ONE. 2011;6(10):e25513. doi: 10.1371/JOURNAL.PONE.0025513
[22] The Chemical Information Ontology [Internet]. San Francisco (CA): GitHub, Inc.; [cited 2025 Mar 19]. Available from: https://github.com/semanticchemistry/semanticchemistry
[23] information-artifact-ontology/IAO [Internet]. San Francisco (CA): GitHub, Inc.; [cited 2025 Mar 19]. Available from: https://github.com/information-artifact-ontology/IAO
[24] Fu G, Batchelor C, Dumontier M, et al. PubChemRDF: towards the semantic annotation of PubChem compound and substance databases. J Cheminform. 2015;7(1):1–15. doi: 10.1186/s13321-015-0084-4
[25] PubChemRDF official documentation [Internet]. Bethesda (MD): PubChem; [cited 2025 Mar 19]. Available from: https://pubchem.ncbi.nlm.nih.gov/docs/rdf
[26] PubChemRDF version 1.8.0 beta, released in Feburary 2023 [Internet]. Bethesda (MD): PubChem; [cited 2025 Mar 19]. Available from: https://pubchem.ncbi.nlm.nih.gov/docs/rdf-release-1-8-0-beta
[27] PubChemRDF model in ShEx [Internet]. Bethesda (MD): PubChem; [cited 2025 Jun 26]. Available from: https://ftp.ncbi.nlm.nih.gov/pubchem/RDF/schema/model.shexc
[28] MaastrichtU-IDS/semanticscience [Internet]. San Francisco (CA): GitHub, Inc.; [cited 2025 Jun 26]. Available from: https://github.com/MaastrichtU-IDS/semanticscience
[29] Validating RDF Data Emulating recursive property paths [Internet]. San Rafael (CA): Morgan & Claypool; [cited 2025 Mar 19]. Available from: https://book.validatingrdf.com/bookHtml010.html#sec104
[30] Shape Expressions (ShEx) Primer, Draft Community Group Report 25 May 2022 [Internet]. Wakefield (MA): World Wide Web Consortium; [cited 2025 Mar 19]. Available from: http://shex.io/primer-next/
[31] Wielemaker J, Schrijvers T, Triska M, et al. SWI-Prolog. Theory Pract Of Log Program. 2012;12(1–2):67–96. doi: 10.1017/S1471068411000494
[32] SWI Prolog [Internet]. [cited 2025 Mar 19]. Available from: https://www.swi-prolog.org/
[33] Pereira FCN, Warren DHD. Definite clause grammars for Language analysis–A survey of the formalism and a comparison with augmented transition networks. Artif Intell. 1980;13(3):231–278. doi: 10.1016/0004-3702(80)90003-X
[34] DCG Grammar rules [Internet]. [cited 2025 Mar 19]. Available from: https://www.swi-prolog.org/pldoc/man?section=DCG
[35] Musen MA. The protégé project: a look back and a look forward. AI Matters. 2015;1(4):4–12. doi: 10.1145/2757001.2757003
[36] ChEBI [Internet]. Hinxton (UK): EMBL-EBI; [cited 2025 Mar 19]. Available from: https://www.ebi.ac.uk/chebi/

Appendices

Appendix A List of prefix/URI correspondences in the PoLyInfoRDF ShEx schema

PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX sio: <http://semanticscience.org/resource/>
PREFIX prism: <http://prismstandard.org/namespaces/1.2/basic/>
PREFIX oboInOwl: <http://www.geneontology.org/formats/oboInOwl#>
PREFIX pli: <http://dice.nims.go.jp/ontology/PoLyInfo-ont/Schema#>
PREFIX homo: <http://dice.nims.go.jp/ontology/PoLyInfo-ont/polyinfo-rdf/Homopolymer#>
PREFIX co: <http://dice.nims.go.jp/ontology/PoLyInfo-ont/polyinfo-rdf/Copolymer#>
PREFIX blend: <http://dice.nims.go.jp/ontology/PoLyInfo-ont/polyinfo-rdf/Blend#>
PREFIX mono: <http://dice.nims.go.jp/ontology/PoLyInfo-ont/polyinfo-rdf/Monomer#>
PREFIX path: <http://dice.nims.go.jp/ontology/PoLyInfo-ont/polyinfo-rdf/Polymerization#>
PREFIX cu: <http://dice.nims.go.jp/ontology/PoLyInfo-ont/polyinfo-rdf/ConstitutionalUnit#>
PREFIX ju: <http://dice.nims.go.jp/ontology/PoLyInfo-ont/polyinfo-rdf/JunctionUnit#>
PREFIX ref: <http://dice.nims.go.jp/ontology/PoLyInfo-ont/polyinfo-rdf/Reference#>

Appendix B Various forms of descriptors and property values

In this appendix, we describe the RDF representations of descriptors and properties

B1 Descriptors

As in Section 3.1, descriptors are attribute values, and they are classified into the following five categories depending on the nature of the values they hold: Their sample RDFs are shown in Figures B1, B2.

(1) Numeric

Numeric descriptors denote numeric values and their RDFs contain numeric literals with or without units. The RDFs have one of the two patterns according to the descriptor class. First, Figure B1(1–1) is an RDF of a single-valued descriptor, where the numeric literal 5.46e + 02 is linked through sio:SIO_000300 (has value), stating that this descriptor has type pli:Wavelength and value 5.46e + 02 with unit obo:UO_0000018 (nanometer). Figure B1(1–2) is an RDF of a range-valued descriptor and the two literals for the range bounds are linked through pli:hasMaxValue and pli:hasMinValue, stating that this descriptor has type pli:Frequency and range bounds of 1.351e + 01 and 5.29e + 00, respectively, with unit obo:UO_0000106 (hertz). Additionally, the optional inequality sign 'ca' means 'circa' indicating that these values are read by humans visually from figures in an original paper, hence they are insufficiently accurate.

(2) String

String descriptors denote string values and their RDFs contain string literals linked through sio:SIO_000300 (has value). Figure B1(2) shows a descriptor that has the type pli:SourceBasedName and string value 'poly(phenanthrene-9,10-diyl)'.

(3) Discrete

Discrete descriptors denote discrete values, a value which is one of the elements of a particular finite set. For example, suppose the set ColorCategory = {red, green, yellow, black, white}; then, red is a discrete value of the ColorCategory type and the color descriptors, which each holds one of these values, are discrete descriptors. Here, we regard a type (ColorCategory in this case) as the set of all values of this type. Note that the color is often denoted by an RGB value, a 3-dimensional continuous value. By specifying a set of color categories that arise from underlying categorization rules for RGB values, a discrete version of color descriptor as above (carrying a value in ColorCategory) is obtained. Similarly in PoLyInfo, discrete values are a kind of categorization of (possibly continuous) observed values. For example, the set of values for the UL flammability code rating type is {V0, V1, V2}, which indicates the categorized degrees of incombustibility observed through flammability tests.

The RDFs have one of the two patterns according to the descriptor class. First, Figure B1(3–1) is the RDF of a descriptor of type pli:SampleType, where its value is the named individual pli:ChelatePolymer. For descriptor classes with this pattern, the underlying finite set is a set of named individuals defined in the ontology such as pli:SampleType = {pli:ChelatePolymer, pli:InorganicPolymer, pli:NeatResin, pli:Composite, pli:Compound}. Figure B1(3–2) is an RDF of a descriptor of type pli:GeneralFeature, where its value is a string literal 'Thermoplastic'. The underlying set is a set of string literals (such as 'Thermoplastic', 'Elastomer', etc.). Some of these literals optionally have an equivalent named individual (pli:Thermoplastic) of the same type (pli:SampleType), which is linked through owl:sameAs. We may say that for a descriptor class in (3–1) the descriptors of this class are, in a sense, 'normalized' (i.e. referred to by named individuals defined in the ontology). For a class in (3–2) only a part (or none) of the descriptors of this class (through owl:sameAs) have named individuals, hence, not 'normalized'.

(4) Descriptive

Descriptive descriptors denote descriptive values, values indirectly specified through a description and not explicitly specified by the actual value. For example, suppose we are specifying the weight value of a person X. Instead of an explicit value such as '70.3 kg', we may specify through a description, such as 'the average weight of Japanese adult men'. The RDFs have a description linked through sio:SIO_000255 (has annotation), which has a free-form comment specifying the value in question. Figure B1(4) illustrates a space group descriptor.

(5) Nested

The nested descriptors denote nested values. For example, the profile of a person consists of the constituent values of their name, gender, and age. Similarly, the nested descriptors consist of other constituent descriptors linked through obo:BFO_0000051 (has part). In PoLyInfoRDF they are used to represent structure descriptors. For example, Figure B2 shows a sample RDF of a primary structure descriptor, a kind of structure descriptor. The primary structure node (A) consists of other descriptors (B)–(E), some of which further comprise constituent descriptors, which forms a tree structure with respect to the parthood relationships.

B2 Property values

Property values were classified into the following three categories according to the pattern of their RDF graphs, as determined by the property class. The sample RDFs of the property value (measurement datum) nodes are shown in Figures B3, B4, together with their property and measurement nodes (typically, in PoLyInfoRDF, these three nodes appear in combination). In the figure, the red dotted rectangles indicate the property value portions with a measurement datum node as the root. Below are brief descriptions of RDF (1), (2), and (3). For more details, refer to the corresponding RDF (1), (3), and (4) in Figure B1.

(1) Numeric

The examples in Figure B3(1) include the numeric values of the UL temperature index time (1–1) and melting temperature (1–2) properties. For (1–1), in the actual RDF graph, the measurement datum nodes for this property do not have a measurement node, which is depicted here to illustrate the general form of the numeric property values specified by the schema.

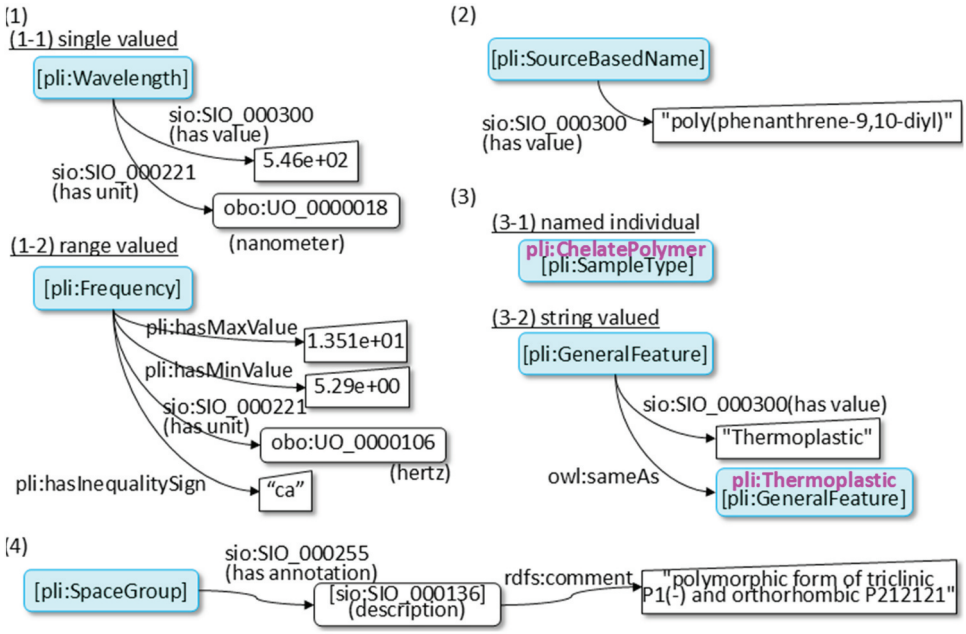


Figure B1. RDF patterns of descriptors. (1) numeric, (2) string, (3) discrete and (4) descriptive.

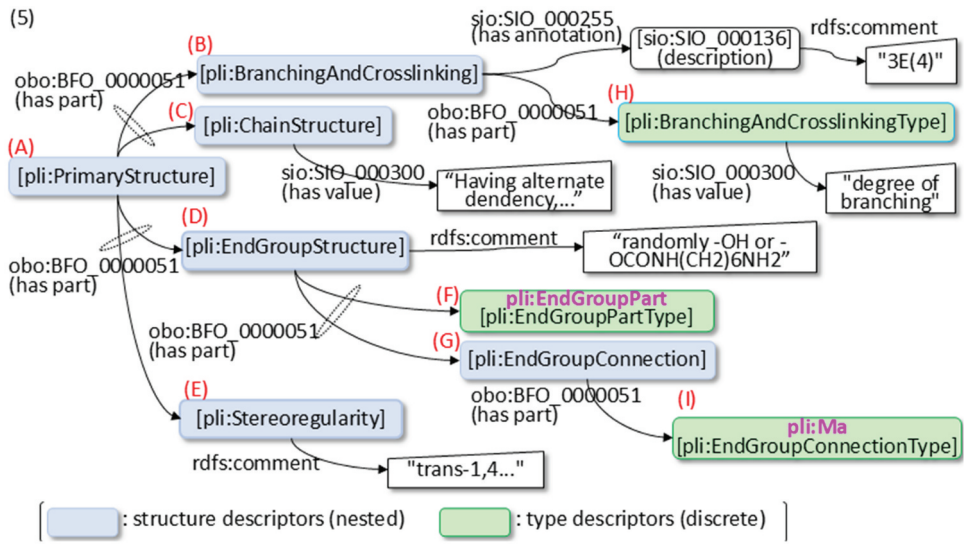


Figure B2. RDF patterns of descriptors. (5) nested.

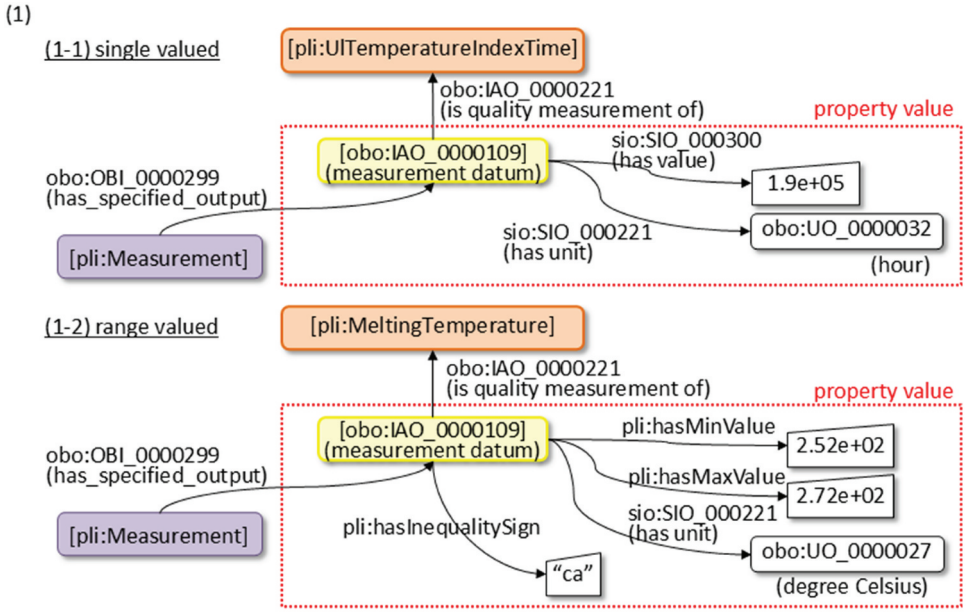


Figure B3. RDF patterns of property values. (1) numeric.

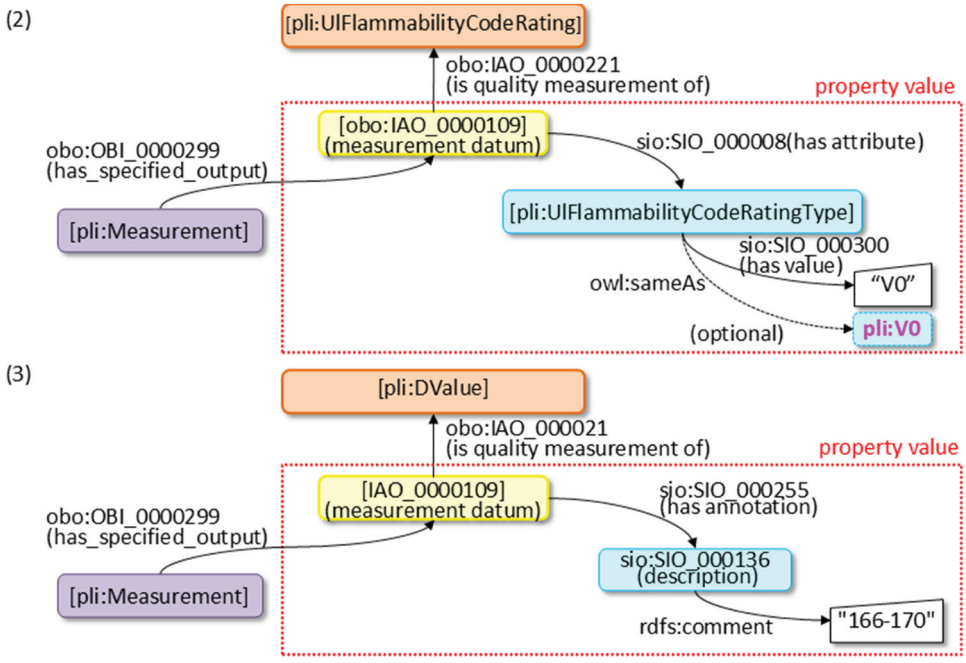


Figure B4. RDF patterns of property values. (2) discrete and (3) descriptive.

(2) Discrete

The example in Figure B4(2) shows a discrete value for the UL flammability code rating. Note that this is parallel to (3–2) in Figure B1, and, to date, (3–1) has not yet been applied to the property values.

(3) Descriptive

The example in Figure B4(3) shows the descriptive value of the d-value properties. This value, which represents the spacing for diffraction, is often not clearly defined in the original paper, and the actual value is not explicitly specified.

Appendix C Example usages of descriptor and property shapes in the PoLyInfoRDF schema

(1) Usage of descriptor shapes

Figure C1a illustrates the ShEx code for the homopolymer nodes as an example of how a non-abstract descriptor shape is used in a schema. The <Homopolymer> shape stipulates a condition on homopolymer nodes in an RDF graph, such that each node has two types pli:Homopolymer and obo:CHEBI_33839 (macroolecule in the Chemical Entities of Biological Interest (ChEBI) ontology [36]), one or more polymer type descriptors, one constitutional unit descriptor, and 0 or more descriptors of the source based name, structure based name, abbreviation, Japanese name, or other name. All these descriptors are linked through sio:SIO_000008 (has attribute). Note that their shapes, such as <PolymerType>, are not defined here but in the imported ShEx files signified by the preceding IMPORT directives, and referenced (reused) here. For reference, an example RDF code for this schema is shown in Figure C1b.

(2) Usage of property shapes

Figure C2a presents the ShEx code for the melting temperature property as an example of how a property shape (in layer3) is defined using abstract shapes. For <PolymerSample> shape

(a)

```
IMPORT <typeDescriptors.shex>
IMPORT <nameDescriptors.shex>
IMPORT <structureDescriptors.shex>

START=@<Homopolymer>

<Homopolymer> CLOSED {
   rdf:type [ pli:Homopolymer ] ;
   rdf:type [ obo:CHEBI_33839 ] ;

   sio:SIO_000008 @<PolymerType> + ;

   sio:SIO_000008 [ cu:~ ] ;    # constitutional unit

   sio:SIO_000008 @<SourceBasedName> OR @<StructureBasedName> OR
@<Abbreviation> OR @<JapaneseName> OR @<OtherName> * ;

   dcterms:identifier xsd:string ;
   rdfs:label xsd:string ?
}
```

(b)

```
homo:P522051 a pli:Homopolymer,
      obo:CHEBI_33839 ;
   rdfs:label "poly(phenanthrene-9,10-diyl)" ;
   dcterms:identifier "P522051" ;
   sio:SIO_000008 pli:OtherPolymers,
      cu:CU522051,
      homo:P522051-1_SourceBasedName,
      homo:P522051-1_StructureBasedName .

homo:P522051-1_SourceBasedName a pli:SourceBasedName ;
   sio:SIO_000300 "polyphenanthrene" .

homo:P522051-1_StructureBasedName a pli:StructureBasedName ;
   sio:SIO_000300 "poly(phenanthrene-9,10-diyl)" .
```

Figure C1. Usage of string descriptors. Examples of (a) ShEx schema and (b) RDF.

(a)

```
IMPORT <propertyBase.shex>

START=@<PolymerSample>

<PolymerSample> CLOSED {
   rdf:type [ pli:PolymerSample ] ;
   sio:SIO_000008 @<Property> +
}

<Property> RESTRICTS @<Property_base> {
   rdf:type [ pli:MeltingTemperature ] ;
}

<IAO_0000109> @<IAO_0000109_base_numeric>

<Measurement> @<Measurement_base>
```

(b)

```
sample:0071313-001-001-002 a pli:PolymerSample ;
   sio:SIO_000008 prop:0071313-001-001-002-0312-001_MeltingTemperature .

prop:0071313-001-001-002-0312-001_MeltingTemperature a pli:MeltingTemperature .

prop:0071313-001-001-002-0312-001_MeltingTemperatureMeasurementDatum a obo:IAO_0000109 ;
   pli:hasInequalitySign "ca" ;
   pli:hasMaxValue 2.72e+02 ;
   pli:hasMinValue 2.52e+02 ;
   obo:IAO_0000221 prop:0071313-001-001-002-0312-001_MeltingTemperature ;
   sio:SIO_000221 obo:UO_0000027 .

prop:0071313-001-001-002-0312-001_MeltingTemperatureMeasurement a pli:Measurement ;
   obo:OBI_0000299 prop:0071313-001-001-002-0312-001_MeltingTemperatureMeasurementDatum ;
   sio:SIO_000008 [ a pli:MeasurementMethod ;
         rdfs:label "DSC" ],
      [ a sio:SIO_000136 ;
         rdfs:comment "Heating rate;20C/min" ] .
```

Figure C2. Usage of numeric properties (melting temperature). Examples of (a) ShEx schema and (b) RDF.

refer to Figure 2. <Property> shape stipulates a condition on melting temperature nodes in an RDF graph, such that the nodes should meet the condition imposed by the <Property_base> shape (due to the RESTRICTS keyword), and further their type should be pli:MeltingTemperature. The <IAO_0000109> and <Measurement> shapes (each for measurement datum and measurement nodes) are equivalent to the <IAO_0000109_base_numeric> (indicating that the values of this property are numeric) and <Measurement_base> shapes, respectively. Most of these shapes are defined in the propertyBase.shex file as signified by the preceding IMPORT directive. For reference, an example of the RDF code for this schema is shown in Figure C2b.

Appendix D Trial code fragments for SPARQL generation

Figure D1a,b show illustrative fragments of the ShEx schema and its corresponding Prolog code in our current trial implementation for SPARQL generation (5.2). These simplified codes are not the same as the actual codes. The schema fragment (a) shows the links among the PolymerSample, Property and IAO_0000109 nodes. In (b), the fragment for the PolymerSample node roughly indicates that the SPARQL string for this _self node is the condition that it has the type pli:PolymerSample. Furthermore, the succeeding link statement shows that this node is linked through sio:SIO_000008 to a Property node of the type specified by _type. The Property fragment states that the SPARQL string for this _self node is the condition of a Property_base node of type _type, followed by the condition of having the type specified by _type. Furthermore, the refers statement shows that the Property shape inherits the Property_base shape, implying that the nodes for these two shapes are identical. These link and refers statements clarify the node configuration in an RDF graph, which is essential for determining the paths requested by a ***path*** command. The same applies to the IAO_0000109 fragments. Note that parameter _type specifies a property class, such as pli:GlassTransitionTemperature, in example user query in Figure 14a.

(a)

```
<PolymerSample> CLOSED {
   rdf:type [ pli:PolymerSample ] ;
   sio:SIO_000008 @<Property> +
}

<Property> RESTRICTS @<Property_base> {
   rdf:type [ pli:{%PROPERTY} ] ;
}
<IAO_0000109> EXTENDS @<NumericValue>
   EXTENDS @<IAO_0000109_base> CLOSED {}

<IAO_0000109_base> {
   rdf:type [ obo:IAO_0000109 ] ;
   obo:IAO_0000221 @<Property_base> ;
   ^obo:OBI_0000299 @<Measurement_base> ?
}
```

(b)

```
% PolymerSample
PolymerSample(_self) -->
        type(_self, pli:PolymerSample).
link(PolymerSample, sio:SIO_000008, Property(_type), [_type, property]).

% Property
Property(_self, _type) -->
        Property_base(_self, _type),
        type(_self, _type) .
refers(Property(_type), Property_base(_type)).

% IAO_0000109(measurement datum)
IAO_0000109(_self, _type) -->
        NumericValue(_self),
        IAO_0000109_base(_self, _type).
refers(IAO_0000109(_type), NumericValue).
refers(IAO_0000109(_type), IAO_0000109_base(_type), [_type, numeric]).

IAO_0000109_base(_self, _type) -->
        type(_self, obo:IAO_0000109).
link(IAO_0000109_base(_type), obo:IAO_0000221, Property_base(_type)).
link(IAO_0000109_base(_type), '^obo:OBI_0000299', Measurement_base(_type)).
```

Figure D1. SPARQL generation from user queries. Examples of (a) ShEx schema and (b) Prolog code (containing DCG grammar rules).