Wikidata Constraints on MARS

David Martin[1] and Peter F. Patel-Schneider[2]

[1] Unaffiliated
[2] Palo Alto Research Center

Abstract. Wikidata constraints, albeit useful, are represented and processed in an incomplete, ad hoc fashion. Constraint declarations do not fully express their meaning, and thus do not provide a precise, unambiguous basis for constraint specification, or a logical foundation for constraint-checking implementations. In prior work we have proposed a logical framework for Wikidata as a whole, based on multi-attributed relational structures (MARS) and related logical languages. In this paper we explain how constraints are handled in the proposed framework, and show that nearly all of Wikidata's existing property constraints can be completely characterized in it, in a natural and economical fashion. We also give characterizations for several proposed property constraints, and show that a variety of non-property constraints can be handled in the same framework.

1 Introduction

Constraints are extremely useful in Wikidata, as they can be in any knowledge base. In Wikidata, property constraints express regularities (patterns of data) which should hold in general, but may have exceptions [12]. In practice, they are used to identify potential problems (constraint violations) to interested contributors who can then either fix the problem or determine that the particular anomaly is acceptable.

One simple example is the symmetric constraint[3] which is understood to indicate that whenever a fact $p(s, o)$[4] exists for a symmetric property p (such as spouse), the fact $p(o, s)$ should normally also be present. As of mid-June 2020 there were over thirty-eight hundred non-symmetric spousal relationships in Wikidata. We know this because of a report generated by a constraint-checking tool. Greater contributor effort, or perhaps additional tools, are needed to determine how many of these non-symmetries are due to missing spouse statements (as opposed to legitimate exceptions), and then create them, but that is a separate challenge. The point here is simply that this constraint-checking tool has produced a valuable report.

Wikidata constraints, however, are represented and processed in an incomplete, ad hoc fashion. Although in most cases they are declared and documented reasonably clearly, the declarations do not fully express their meaning. For example, it is possible to declare that spouse is subject to the symmetric constraint. However, crucially,

[3] For readability we use the English label to identify a Wikidata item, here `https://www.wikidata.org/wiki/Q21510862`. In formulas, we replace spaces with underscores.
[4] We use *predicate(subject, object)* notation rather than *(subject, predicate, object)*.

there is no formal characterization of what it *means* for a property to be symmetric. That is only stated in natural language documentation.

Stepping outside of Wikidata, it is straightforward to formally express this meaning in first-order logic (FOL) (with x and y as free variables, as explained in Section 3):

$$\mathsf{spouse}(x, y) \rightarrow \mathsf{spouse}(y, x) \tag{1}$$

The value of formal characterizations is foundational in Computer Science. We rely on them for clarity in specification in most of our activities. And yet Wikidata lacks the logical framework to take advantage of characterizations like Formula (1). Such a framework, if available, would provide a precise basis for constraint specification, and a logical foundation for constraint-checking implementations.

Further, in current practice specifying a new constraint, and building a constraint checker for it, may be unnecessarily laborious, idiosyncratic, and error-prone. A logical formulation and implementation of constraints would permit constraints to be quickly specified and reduce the implementation burden for each new type of constraint.

In prior work [9], building on the work of Marx et al. [6], we have proposed a logical framework for Wikidata, which supports the specification of rules that can be used to draw inferences to achieve a much more complete collection of facts, which in turn can support a more comprehensive, effective, and easy-to-use query service over Wikidata. This is done in a way that accounts for, leverages, and facilitates the use of the representational conventions in Wikidata.

Our logical framework also encompasses the handling of constraints. In this paper, we describe how this is done, and show that nearly all of Wikidata's existing property constraints can be given a complete characterization in a natural and economical fashion, using a familiar style of logical expression. These logical formulae, unlike documentation in natural language, provide an unambiguous basis for understanding constraints and for implementing constraint checkers. (Indeed, once an evaluation capability exists for these formulae, checking a new constraint requires no new engineering effort.) We also give characterizations for several proposed property constraints that could usefully be added to Wikidata. In addition, we show that our approach allows for representing and handling a broader range of constraints, going beyond property constraints, in the same formalism.

In the next two sections, we give a general overview of the current handling of constraints in Wikidata, and an overview of our logical framework for Wikidata. In three sections after that, we give examples of our approach's characterization of existing property constraints, proposed property constraints, and several useful non-property constraints. We follow that with discussion, related work, and conclusion sections.

2 Property Constraints in Wikidata

In current Wikidata practice, "constraint" is used for both "property constraints" and "complex constraints". We give here an overview of Wikidata property constraints. Complex constraints[5] (also known as "custom constraints") are not considered in this paper, although our approach can handle constraints beyond property constraints.

[5] https://www.wikidata.org/wiki/Template:Complex_constraint

At present there are 30 property constraint types used in Wikidata, as revealed by the "up-to-date list" SPARQL query link included on [12]. As explained on that page, "constraints for a property are specified as statements on the property, using property constraint (P2302) and the constraint type item". For example, in the notation we've adopted for this paper the following statement says that spouse (P26) is constrained by the symmetric constraint (Q21510862) constraint type.

$$\text{property\_constraint}(\text{spouse}, \text{symmetric\_constraint})$$

Many constraints are configurable by specifying values for parameters, which are stated as qualifiers on the constraint statement. (Statement and qualifier in Wikidata are defined in the Wikibase Data Model [7]). There are several general parameters that can be added to any constraint statement, such as constraint status (which can have values mandatory constraint or suggestion constraint) and exception to constraint (which is used to list known exceptions). There are other parameters that are specific to a particular constraint type, or a small group of constraint types. We shall see examples of some of these in subsequent sections.

3 Logical Framework

Our logical framework for Wikidata [9] supports the use of both rules and constraints. Rules are used to draw inferences; constraints are used to detect the presence of questionable data patterns. After briefly reviewing the prior work of Marx et al. [6] – which produced MARS, MAPL, and MARPL – we then introduce our extensions to these – eMARS, eMAPL, and eMARPL – which are the logical foundations of our approach. In our approach, rules are expressed in eMARPL, and constraints in eMAPL.

MARS, MAPL, and MARPL. As noted in [6], Wikidata's custom data model goes beyond the *Property Graph* data model, which associates sets of attribute-value pairs with the nodes and edges of a directed graph, by allowing for attributes with multiple values. Marx et al. refer to such generalized Property Graphs as *multi-attributed graphs*, and observes that "In spite of the huge practical significance of these data models ..., there is practically no support for using such data in knowledge representation". Given that motivation, Marx et al. introduce the *multi-attributed relational structure* (MARS) to provide a formal data model for generalized Property Graphs, and *multi-attributed predicate logic* (MAPL) for modeling knowledge over such structures. MARS and MAPL may be viewed as extensions of FOL to support the use of attributes (with multiple values).

The essential new elements over FOL are these:

- a *set term* is either a set variable or a set of attribute-value pairs $\{a_1 : b_1, ..., a_n : b_n\}$, where a_i, b_i can be *object terms*. Object terms are the usual basic terms of FOL, and can be either constants or *object variables*.
- a *relational atom* is an expression $p(t_1, ...t_n)@S$, where p is an *n-ary* predicate, $t_1, ...t_n$ are object terms and S is a set term.
- a *set atom* is an expression $(a : b) \in S$, where a, b are object terms and S a set term.

These elements are best illustrated with a simple example (taken directly from [6]):

$$\begin{aligned} \forall x, y, z_1, z_2, z_3.\mathsf{spouse}(x, y)@\{\mathsf{start} : z_1, \mathsf{loc} : z_2, \mathsf{end} : z_3\} \\ \rightarrow \mathsf{spouse}(y, x)@\{\mathsf{start} : z_1, \mathsf{loc} : z_2, \mathsf{end} : z_3\} \end{aligned} \tag{2}$$

This MAPL formula states that spouse is a symmetric relation, where the inverse statement has the same start date, end date, and location. Each occurrence of $\mathsf{spouse}(...)$ $@\{...\}$ is a relational atom, which includes the set term $\{\mathsf{start} : z_1, \mathsf{loc} : z_2, \mathsf{end} : z_3\}$. If that set term were represented by a set variable U, then one could make an assertion about its membership using the set atom $(\mathsf{start} : z_1) \in U$.

In Wikidata terms, this particular relational atom (once x and y have been instantiated to specific Wikidata items) corresponds to a statement, and each attribute-value pair (once the z_i variable has been instantiated to a specific value), corresponds to a qualifier of the statement. (x, of course, is called the subject of the statement, and y the value or object of the statement.) While MAPL allows for predicates of arbitrary arity, in Wikidata all statements are triples; i.e. Wikidata properties have arity 2.

Marx et al. go on to introduce multi-attributed rule-based predicate logic (MARPL), a MAPL fragment which is decidable for fact entailment, but still provides a high level of expressivity. Note that Formula 2 falls within the MARPL fragment. MARPL also allows for a special type of function that can be used to construct an attribute set in the head of a rule. A *MARPL ontology*, then, includes a set of rules and a set of these function definitions. Because the representation and checking of *constraints* in our framework builds on MAPL rather than MARPL, we omit any further details about MARPL.

eMARS, eMAPL, and eMARPL. MARS / MAPL / MARPL are close to providing a logical basis for Wikidata but are still missing two essential elements:

- *Wikidata-specific datatypes.* Datatypes play a large role in Wikidata, and it has its own set of datatypes with certain idiosyncrasies, as documented in [7]. In order to specify the manipulation of data elements in rules, functions and relations are needed for constructing, accessing, and combining the data elements of each of Wikidata's datatypes.
- *A feasible means of specifying the uses of attributes in rules.* Handling Wikidata qualifiers (which are represented as attributes in MAPL and MARPL) correctly requires accounting for potentially many attributes in each of many rules, which is infeasible, from a practical perspective, with MARPL.

In [9], we provide a semi-detailed sketch for addressing each of these needs. (A more formal specification will be provided in a future publication.) Specifically, we define an *extended MARS* (eMARS) as a MARS extended with a specification of datatypes, with their associated relations and functions, and we discuss the functions and relations that are needed for each of Wikidata's datatypes. We define *extended MAPL* (eMAPL) to include *eMAPL terms*, which are MAPL terms augmented with datatype function applications, and *eMAPL formulae*, which allow for the use of eMAPL terms and datatype relations as predicates. To further support the representation of constraints, we also add equality and, as syntactic sugar, counting quantifiers.

To address the second need mentioned above, we introduce *attribute characterizations*, which provide a means to describe the desired behavior of attributes when

rules fire, separately from the rules themselves, and we define an *extended MARPL (eMARPL) ontology* to include, in addition to rules and function definitions, a set of attribute characterizations. We also describe how these characterizations can be used as macros, modifying the functions and rules of an eMARPL ontology.

Given these logical constructs, we show in [9] how Wikidata itself can be represented as an eMARS, and discuss some of the essential rules that are needed for inferencing in Wikidata (including, but not limited to, ontological rules that axiomatize foundational Wikidata concepts such as instance of, subclass of, subproperty of, reflexive property, and transitive property). Other types of rules are possible and important, such as the rules instantiated in the SQID tool [5]. The "meaning of Wikidata" is then the inferential closure of the eMARS under an eMARPL ontology composed of rules, function definitions, and attribute characterizations. It is this eMARS that is used when querying or otherwise requesting what is true in Wikidata, or checking constraints.

Representing Constraints in eMAPL. We model Wikidata constraints as eMAPL formulae that are evaluated over the eMARS that is the "meaning of Wikidata". Because constraint formulae are used as queries, and not for inferencing, we can take advantage of the greater expressiveness of eMAPL. It is known that the data complexity of evaluating FOL formulae is polynomial, and that remains true for eMAPL formulae.

Constraints can either be given a positive formulation, which expresses a pattern of data elements that conform to the constraint, or a negative formulation, which expresses a pattern of data elements that violate the constraint. In our view, it is most natural to first write the positive formulation, and from that derive the negative formulation, which can then be used as a query. (The derivation of the negative formulation starts with applying the negation operator to the positive formulation, and then applies transformations, if desired, based upon well-known laws of logic.)

For example, the distinct_values_constraint in Wikidata indicates that a given property should have different values for different items (across all of Wikidata). The following eMAPL formula[6] embodies this constraint. Here, because we are treating these formulae as queries, the variables are considered to be free variables. We omit attribute sets wherever they are irrelevant to the meaning of the constraint. In other words, for each atom missing an attribute set there is an implicit variable, which can be ignored by a constraint-checker (formula evaluator), or treated as an additional free variable.

$$\begin{aligned} &\mathsf{property\_constraint}(p, \mathsf{distinct\_values\_constraint}) \\ &\quad \wedge p(s1, o1) \wedge p(s2, o2) \wedge s1 \neq s2 \rightarrow o1 \neq o2 \end{aligned} \tag{3}$$

Formula 3 (the positive formulation) directly expresses the meaning of the constraint in the usual fashion of first-order logic. If satisfied (for all possible bindings of the free variables), the constraint has no violations.

In all of the formulae for existing property constraints, we employ Wikidata's property constraint declarations, which works nicely. For example, in Formula 3, the first conjunct will match against one of Wikidata's existing property constraint declarations, thereby binding p to one of the properties having the distinct values constraint (e.g., the ISBN-13 property, P212).

[6] In future work, we plan to develop a keyboard-friendly syntax, requiring no specialized math or logic symbols.

Formula 4 below (the negative formulation), where satisfied, identifies items that violate the constraint.

$$\begin{aligned}&\mathsf{property\_constraint}(p, \mathsf{distinct\_values\_constraint})\\ &\quad \wedge p(s1, o1) \wedge p(s2, o2) \wedge s1 \neq s2 \wedge o1 = o2\end{aligned} \tag{4}$$

Because, in our framework, constraints are checked after the KB has been augmented by running the rules (i.e., the constraints are checked over the "meaning of Wikidata" KB), a far more useful set of results will be obtained. Inferences from rules will instantiate facts that were missing from the original KB, thus providing a complete (with respect to the rules) set of facts to be checked. Consequently, a complete and accurate set of constraint violations will be found, and false positives and negatives (which would have resulted from missing facts) will be avoided.

In our framework, as illustrated above, the specification of a new property constraint type involves, in addition to the creation of property constraint type declarations of the sort used in current practice, an eMAPL formula for the new type (or several formulae, if preferred, in some cases). These formulae, unlike documentation in natural language, provide an unambiguous basis for understanding and implementing constraint checkers. Once an evaluation capability exists for eMAPL formulae, checking a new constraint will require no new engineering effort.

We investigated the extent to which Wikidata's existing property constraints can be expressed in eMAPL, and reported our results in [4]. (This paper is a shortened version of that report.) Out of 26 property constraints examined, only one could not readily be expressed in eMAPL. We also became aware of one *proposed* property constraint that cannot readily be expressed. In both cases, the problem can be addressed in a straightforward manner.

In the next two sections, we show examples of existing and proposed property constraints, expressed in eMAPL, which illustrate more of its features. eMAPL allows for representing and handling a broad range of constraints, going beyond property constraints, in the same formalism. In Section 6, we illustrate this with several examples of non-property constraints. In Section 7, we discuss the two property constraints that could not readily be expressed.

4 Existing Property Constraints

In [4], we give complete characterizations for 26 of the 30 property constraint types in current use. As explained there, we omitted four constraint types – the same four omitted in [1] – due to insufficient documentation being available for them. Here, we present two of the 26 characterizations, to illustrate other features of eMAPL.

The **required qualifier constraint (Q21510856)** provides a nice illustration of attribute set variables and set atoms (from Section 3) in the characterization of a constraint type. Here, we see the set atom $(\mathsf{property} : q) \in CQ$ used to obtain the value q of the property qualifier. q identifies another qualifier whose use is required with the given property. For example, this constraint type is used with the property population (P1082). If this formula were to be evaluated, when p binds with that property, q will bind with the qualifier point in time (P585), which is the "required" qualifier. $p(s, o)@SQ$ will bind

with a fact with property population, and with statement qualifiers SQ. The right-hand side of the formula, then, checks that SQ contains the required qualifier.

$$\begin{aligned}&\mathsf{property\_constraint}(p, \mathsf{required\_qualifier\_constraint})@CQ\\ &\wedge(\mathsf{property} : q) \in CQ \wedge p(s,o)@SQ \rightarrow \exists v.(q:v) \in SQ\end{aligned} \tag{5}$$

This is the positive formulation for this constraint type. As noted above, in practice one would derive and use the negative formulation to identify violations.

The **value type constraint (Q21510865)**, which states that each value of the given property should have a given type (which is also known as the *range* of the property) is an example where it is convenient to express the constraint type with multiple formulae. In this case, we use three formulae – one for each possible value of the relation qualifier (although it could be done with a single formula if desired). The relation qualifier characterizes the allowed relationship between the value and the type (which is given by the class qualifier). Note also that these formulae allow for any number of values for the class qualifier, in keeping with current practice.

$$\begin{aligned}&\mathsf{property\_constraint}(p, \mathsf{value\_type\_constraint})@CQ\\ &\wedge (\mathsf{relation} : \mathsf{instance\_of}) \in CQ \wedge p(s,o)\\ &\quad\rightarrow \exists c.((\mathsf{class} : c) \in CQ \wedge \mathsf{instance\_of}(o,c))\\ &\mathsf{property\_constraint}(p, \mathsf{value\_type\_constraint})@CQ\\ &\wedge (\mathsf{relation} : \mathsf{subclass\_of}) \in CQ \wedge p(s,o)\\ &\quad\rightarrow \exists c.((\mathsf{class} : c) \in CQ \wedge \mathsf{subclass\_of}(o,c))\\ &\mathsf{property\_constraint}(p, \mathsf{value\_type\_constraint})@CQ\\ &\wedge (\mathsf{relation} : \mathsf{instance\_or\_subclass\_of}) \in CQ \wedge p(s,o)\\ &\quad\rightarrow \exists c.((\mathsf{class} : c) \in CQ \wedge (\mathsf{instance\_of}(o,c) \vee \mathsf{subclass\_of}(o,c)))\end{aligned}$$

5 Proposed Property Constraints

Here, we show three other property constraint types that we believe should be included in Wikidata. There are many other useful property constraint types that could be characterized using eMAPL, including many of the suggested types (determined by survey of active Wikipedia editors) listed in [1].

Asymmetric property constraint. Although there is a class asymmetric Wikidata property, there is no property constraint for asymmetry. (This differs from the case of the class symmetric property, which does have a corresponding property constraint.) In any case, the concept of asymmetric property cannot be expressed in eMARPL (and thus, unlike the case of symmetric property, cannot be expressed as a rule of inference). However, asymmetry can easily be expressed as a constraint in eMAPL, as follows.

$$\mathsf{asymmetric\_property}(p) \wedge p(y,x) \rightarrow \neg p(x,y) \tag{6}$$

Local value type constraint. The concept of a "local" value type constraint has proven to be valuable in ontology engineering (where it is sometimes called a "local

range restriction") , and can easily be expressed by extending the characterization of value type constraint (see Section 4). "Local" in this context indicates that the constraint holds when the subject of a statement has a particular type, such as the children of humans being humans. This constraint can be characterized as follows: *If the subject item of a statement has the given type (indicated using qualifier local_class), the referenced (object) item should be a subclass or instance of the given type (indicated using qualifier class).* This constraint calls for a distinct property constraint statement for each local class that one desires to distinguish for a given property (but it's already accepted practice to have multiple property constraint statements for a given property and constraint type). To save space, here we omit the formulae for the instance_or_subclass_of and subclass_of values of relation.

$$\begin{aligned}&\mathsf{property\_constraint}(p, \mathsf{local\_value\_type\_constraint})@CQ \wedge (\mathsf{local\_class} : lc) \in CQ \\ &\wedge (\mathsf{relation} : \mathsf{instance\_of}) \in CQ \wedge p(s,o) \wedge \mathsf{instance\_of}(s, lc) \\ &\quad \rightarrow \exists c.((\mathsf{class} : c) \in CQ \wedge \mathsf{instance\_of}(o, c))\end{aligned}$$

Essential property constraint. The importance of a particular property for items of a particular type could be indicated in a similar fashion to local value type constraint. For example, it would be useful to indicate that a person should normally have a parent property statement. Because there are persons whose parents are unknown, a constraint would be more appropriate for this sort of example than a rule, in our framework. This constraint would provide stronger guidance regarding the importance of a particular property than the existing meta-property properties for this type, which merely indicates the properties that are normally used with items of a particular type. Note that the meaning of this constraint is different than that of allowed entity type constraint, and item requires statement constraint.

This property is also "local" in the sense that it is conditioned on the subject of a statement being of a particular type. In the world of ontology engineering, this constraint is sometimes called a "local existential restriction".

$$\begin{aligned}&\mathsf{property\_constraint}(p, \mathsf{essential\_property\_constraint})@CQ \\ &\wedge(\mathsf{local\_class} : lc) \in CQ \wedge \mathsf{instance\_of}(s, lc) \rightarrow \exists o.p(s,o)\end{aligned} \tag{7}$$

6 Non-Property Constraints

It is natural to consider a broader range of constraints, and desirable to express them all in the same logical framework. Here, we show eMAPL formulae for several useful constraints that fall outside the definition of "property constraint". As noted below, some of these are already present in Wikidata (in some other form besides a constraint). For those that are already present, we leverage the existing Wikidata declarations (as we have done for property constraints). To the best of our knowledge, in current Wikidata practice these examples would normally be checked by creating a bot, which would require a greater effort than simply evaluating one of these formulae (as could be done in our proposed framework), and the effort would likely be relatively ad hoc, cumbersome, and error-prone.

6.1 Union of Classes and Disjoint Classes

The existing union of and disjoint union of (meta-)properties can each be expressed with a pair of formulae. Here, we use the "dummy value" list_values_as_qualifiers (Q23766486) with of (P642), in accord with existing practice for these properties.

$$\begin{aligned}
&\mathsf{union_of}(u, \mathsf{list_values_as_qualifiers})@Q \wedge \mathsf{instance_of}(i, u) \\
&\quad \rightarrow \exists c.((\mathsf{of} : c) \in Q \wedge \mathsf{instance_of}(i, c)) \\
&\mathsf{union_of}(u, \mathsf{list_values_as_qualifiers})@Q \wedge (\mathsf{of} : c) \in Q \wedge \mathsf{instance_of}(i, c) \\
&\quad \rightarrow \mathsf{instance_of}(i, u) \\
&\mathsf{disjoint_union_of}(u, \mathsf{list_values_as_qualifiers})@Q \wedge \mathsf{instance_of}(i, u) \\
&\rightarrow \exists c1.((\mathsf{of} : c1) \in Q \wedge \mathsf{instance_of}(i, c1) \\
&\quad \wedge \forall c2.(((\mathsf{of} : c2) \in Q \wedge \mathsf{instance_of}(i, c2)) \rightarrow c1 = c2)) \\
&\mathsf{disjoint_union_of}(u, \mathsf{list_values_as_qualifiers})@Q \wedge (\mathsf{of} : c) \in Q \wedge \mathsf{instance_of}(i, c) \\
&\rightarrow \mathsf{instance_of}(i, u)
\end{aligned}$$

disjoint with, a proposed property, was discussed in 2016 but not adopted. In our opinion, it would be a valuable addition to Wikidata.

$$\mathsf{disjoint\_with}(c1, c2) \rightarrow \neg\exists i.(\mathsf{instance\_of}(i, c1) \wedge \mathsf{instance\_of}(i, c2))$$

6.2 No-value Constraint

We think the best treatment of a no-value snak[7] is as a constraint but it is unclear whether a no-value snak means no value at all, no value with the same qualifiers (as the no-value snak), or something in between. These options can be modelled as eMAPL constraint formulae. Note that the some-value snak doesn't call for a constraint, but is addressed by other means in [9].

Formula 8 captures the "no value at all" interpretation. Note that $\mathsf{no\_value}(p, s)$ statements do not exist *per se* in Wikidata, but could be generated from Wikidata's internal representation of PropertyNoValueSnak.

$$\mathsf{no\_value}(p, s) \rightarrow \neg\exists o.p(s, o) \tag{8}$$

Formula 9 captures the "no value with same qualifiers" interpretation.

$$\mathsf{no\_value}(p, s)@Q \rightarrow \neg\exists o.p(s, o)@Q \tag{9}$$

6.3 Other Examples

Formula 10 expresses the existing *metasubclass of* relation between two metaclasses: instances of the metaclass $m1$ are likely to be subclasses of classes that are instances of the metaclass $m2$.

$$\begin{aligned}
&\mathsf{metasubclass_of}(m1, m2) \wedge \ \mathsf{instance_of}(c1, m1) \rightarrow \\
&\quad \exists c2.(\mathsf{subclass_of}(c1, c2) \wedge \mathsf{instance_of}(c2, m2))
\end{aligned} \tag{10}$$

[7] https://www.mediawiki.org/wiki/Wikibase/DataModel#PropertyNoValueSnak

Formula 11 states that no item should be both instance of and subclass of the same other item. Formula 12 disallows loops in *subclass of* hierarchies. Neither of these useful constraints, to our knowledge, are currently declared or checked in Wikidata.

$$\text{instance\_of}(i1, i2) \rightarrow \neg\text{subclass\_of}(i1, i2) \tag{11}$$

$$\text{subclass\_of}(c1, c2) \wedge c1 \neq c2 \rightarrow \neg\text{subclass\_of}(c2, c1) \tag{12}$$

7 Discussion

In a setting such as our proposed framework, there are some logical characterizations that can be sensibly used as either rules or constraints. For example, the concept of *symmetric property*, treated as a property constraint in Wikidata and thus included as a constraint in this paper, could be used as a rule in our framework, if one considers that it has no exceptions. We tend towards this view ourselves, and in fact, offer a rule for symmetric property in [9], as well as rules that characterize the meaning of reflexive property, transitive property, instance of, subclass of, and subproperty of. In our framework, if a logical characterization is considered to be without exception, and can be expressed in eMARPL, there is no need to express it as a constraint. This is because the reasoning provided by firing the rule will ensure that there are no exceptions to be found by a constraint formula.

Some constraints (any whose eMAPL formula is also an eMARPL rule) could be used as rules, as-is. Given a framework that allows for both rules and constraints, such as our proposed framework, it isn't necessarily obvious in every case whether a logical characterization should be treated as a rule or a constraint. It can depend not only on logical expressiveness, but also on intuitions and practices that have developed in the community. For example, the authors' intuition and experience indicate that the concept of symmetry is inherent in symmetric properties *by definition* (as can easily be seen in the case of spouse or sibling), and thus one needn't and shouldn't allow for exceptions. Space constraints preclude a full discussion of this question of whether a rule or constraint usage is more suitable for a given logical characterization.

In current Wikidata practice, there is evidence of considerable ambivalence about the extent to which property constraints should allow for exceptions. The Help page for property constraints [12] states that "Constraints are hints, not firm restrictions, and are meant as a help or guidance to the editor. They can have exceptions...". At the same time, any constraint can be marked with a constraint status of mandatory, and 29.2% of constraints are characterized in this way, whereas only 4.6% of constraints have specified allowed exceptions (using the exception_to_constraint qualifier) [1]. Moreover, the "Wikidata:2020 report on Property constraints" [1] lists as a goal (i.e., an "ideal state") for 21 existing property constraint types that they should have no exceptions (e.g.,"Goal: No value type constraint on Wikidata has exceptions.").

We believe this ambivalence exists, in part, because Wikidata doesn't currently provide an effective representation of rules (or a mechanism for deriving inferences from them), and thus the existing constraints framework has been forced to accommodate

some things that ought to be rules (symmetric property, etc.). This provides another strong argument for the adoption of a framework such as ours.

In our framework, because of their use in reasoning, the expressiveness of rules necessarily must be more limited than that of constraint formulae. Thus, there are a few useful logical characterizations (e.g., *union of*, *disjoint union of*, *disjoint classes*) that one might wish to expresses as rules, but would not be able to. In such cases, it would be perfectly reasonable to check them as constraints. If desired, one could arrange by various means to ensure that violations of these constraints are not allowed to occur, thus achieving the effect of a rule, albeit in a somewhat more cumbersome fashion.

We identified one existing constraint (the Commons link constraint) and also became aware of one proposed property constraint (acyclicity) that cannot be readily expressed in eMAPL. The Commons link constraint requires knowledge that is not contained in Wikidata. However, by adding Wikimedia Commons metadata to Wikidata (one fact per WC page, giving its name and namespace), this constraint can be easily expressed. Additional details are available in [4]. The proposed *acyclic* property constraint, mentioned in [1], would check whether a property's usage has caused a cycle (e.g., A is B's mother, B is C's mother, C is A's mother), which is outside the expressiveness of an FOL-based logic. However, because eMAPL is used only as a query language, it could be extended with property path constructs, like those of SPARQL [11], which would allow for the expression of this proposed constraint.

We have not yet encountered any desirable *non-property* constraints that could not be expressed; however, we have not yet performed a thorough search for candidate non-property constraints.

8 Related Work

While there isn't space to survey the large literature of logical frameworks for knowledge bases, we can highlight relevant work from several slices of that literature.

SPARQL. SPARQL is used extensively with Wikidata, via the Wikidata RDF dump, and in some constraint checking is used in somewhat the same way as we envision for eMAPL. Indeed, translation to SPARQL would be one implementation option for handling constraints expressed in eMAPL. SPARQL, of course, supports filters and many other expressiveness features. However, as noted in Section 7, so far we've only identified one proposed constraint (acyclicity) that goes beyond the expressiveness of eMAPL—and eMAPL could be extended in a well-understood manner to allow for this.

SPARQL also has the advantage of being supported by many existing products. However, eMAPL provides an attribute set notation for qualifiers, which is far more natural and readable than using SPARQL over the complex representation of qualifiers in the RDF dump. Similarly, eMAPL provides Wikidata-specific datatype functions and relations, which, again, results in simpler, more natural, more compact expressions in some cases. eMAPL allows for deployment options that are more integral with the native deployment of Wikidata, thus removing dependency on the RDF dump, and potentially allowing for more continuous, up-to-date constraint checking. At the same time, eMAPL provides a logical foundation for a broader array of deployment options that are external to Wikidata's native deployment.

Constraints in KBs. Wikidata's (and our) perspective on constraints is consistent with the view taken by other recent work on constraints for knowledge-graph-like systems. The SHACL Shapes Constraint Language [3], a W3C Recommendation since July 2017, and the Shape Expressions Language 2.1 (ShEx) [10] are each used to specify valid data patterns in RDF KBs, and provide a framework for identifying violations of those patterns. (ShEx is used in WikiProject Schemas[8]). The primary differences from our approach are that they are RDF-specific, and are grounded in pattern matching techniques rather than in evaluation of logical formulas. In addition, our approach provides support for Wikidata-specific data types and Wikidata's use of qualifiers, and benefits from its role in a larger logical framework that supports rule-based inference. [8] shows how Description Logic axioms (when interpreted in a closed-world setting) can be used for constraint checking, discusses their applicability to RDF KBs, and shows the feasibility of translation to SPARQL as an implementation strategy. The approach herein builds on FOL rather than Description Logic, and again, addresses challenges specific to Wikidata.

Logical foundations for Wikidata. SQID [5] is a browser and editor for Wikidata, which draws inferences from a collection of MARPL rules. Our work was informed by SQID's embodiment of MARPL-based reasoning, and motivated in part by the desire to expand the expressiveness of MARPL rules, as illustrated by the SQID rule set to provide a more complete reasoning framework, and to accommodate Wikidata constraints. [2] also formalizes a model of Wikidata based on MARS, but with a different objective: the application of "Formal Concept Analysis to efficiently identify comprehensible implications that are implicitly present in the data". [2] is thus nicely complementary with [6] and with our work, in that it provides a basis for discovering, rather than hand-authoring, new (e)MARPL rules.

9 Conclusion and Future Work

After reviewing our prior work that proposes a logical framework for Wikidata, based in part on extended multi-attributed predicate logic (eMAPL), we showed how the framework can be used to give logical characterizations (eMAPL formulas) for constraints in Wikidata, in a manner that makes use of Wikidata's existing constraint declarations, but goes beyond them to give a complete expression of their meaning. We explained, at a high level, how constraint checking would take place in our framework. We are only aware of two property constraints (one existing, one proposed) which cannot currently be expressed in eMAPL; we explained how these could be addressed with extensions (to Wikidata content in one case, eMAPL in the other). Characterizations are also given for several proposed property constraints, and for several non-property constraints whose use could benefit Wikidata.

In future work, we plan to develop a detailed design for a scalable deployment of our proposed logical framework, in a manner that could integrate well with existing Wikidata infrastructure, workflow, and practices. We also plan to give eMAPL characterizations of the suggested property constraint types (determined by survey of active Wikipedia editors) in [1], and analyze Wikidata's existing complex constraints and the degree to which they could be accommodated in our framework.

[8] https://www.wikidata.org/wiki/Wikidata:WikiProject_Schemas

References

1. Abin, D.: Wikidata:2020 report on Property constraints (2020 (accessed August 6, 2020)), https://www.wikidata.org/wiki/Wikidata:2020_report_on_Property_constraints
2. Hanika, T., Marx, M., Stumme, G.: Discovering implicational knowledge in Wikidata. In: International Conference on Formal Concept Analysis. pp. 315–323. Springer (2019)
3. Knublauch, H., Kontokostas, D.: Shapes constraint language (SHACL). W3C Recommendation, http://www.w3.org/TR/shacl/ (2017)
4. Martin, D.L., Patel-Schneider, P.F.: Wikidata constraints on MARS (extended technical report). CoRR **abs/2008.03900** (2020), https://arxiv.org/abs/2008.03900
5. Marx, M., Krötzsch, M.: SQID: Towards ontological reasoning for Wikidata. In: International Semantic Web Conference (Posters, Demos & Industry Tracks) (2017)
6. Marx, M., Krötzsch, M., Thost, V.: Logic on MARS: Ontologies for generalised property graphs. In: IJCAI. pp. 1188–1194 (2017)
7. MediaWiki: Wikibase/DataModel (accessed August 6, 2020), https://www.mediawiki.org/wiki/Wikibase/DataModel
8. Patel-Schneider, P.F.: Using description logics for RDF constraint checking and closed-world recognition. CoRR **abs/1411.4156** (2014), http://arxiv.org/abs/1411.4156
9. Patel-Schneider, P.F., Martin, D.: Wikidata on MARS. In: Proceedings of the 33rd International Workshop on Description Logics (September 2020)
10. Prud'hommeaux, E., Boneva, I., Gayo, J.E.L., Kellogg, G.: Shape expressions: An RDF validation and transformation language. W3C Final Community Report, http://shex.io/shex-semantics/ (October 2019)
11. SPARQL 1.1 query language. W3C Recommendation, http://www.w3.org/TR/sparql11-query/ (2013)
12. Wikidata.org: Help:Property constraints portal (accessed August 6, 2020), https://www.wikidata.org/wiki/Help:Property_constraints_portal