Ontology Completeness and Connectedness – a Generic Evaluation Model

Philippe A. MARTIN [a,1]
[a] *EA2525 LIM, University of La Réunion, F-97490 Sainte Clotilde, France*

Abstract. A classic general definition of the completeness of an ontology – or, more generally, a knowledge base (KB) – is: the degree to which information required to satisfy a specification are present in this KB. Most current completeness measures are metrics about relatively how many objects from a reference KB are represented in the evaluated KB. There also are many tools implementing in *ad hoc* ways particular measures for evaluating the degree to which a KB complies with particular design rules (ontology patterns, best practices, methodologies, ...). The present article shows how the "semantic connectedness" notion – i.e., (relatively) how many objects in a KB satisfy a semantic specification – can be precisely defined for specifying such measures and implementing or extending them in a generic way. These completeness measures are not of the above cited kind (they do not require a reference KB) and they allow the specification of KB quality measures not usually categorized as completeness measures. This article introduces and defines various semantic connectedness notions, via various functions (and various kinds of parameters) which all exploit the same general idea. This idea and definitions are the main contributions of this article since they answer several important research questions which can be merged into the following one: how to define semantic connectedness in a way that is generic, automatically checkable and that supports the writing of ontology design rules generalizing current ones and better leading to the increase or maximization of the entering of particular relations and then of inferences from them. As an illustration of the implemented tool and validations made for the introduced approach, this article also shows i) one interface of the tool, displaying interesting types of relations and parameters to use for checking semantic connectednesses, and ii) some results of the evaluation of a well known foundational ontology.

Keywords. ontology completeness, knowledge organization, ontology evaluation

1. Introduction

Dataset completeness. As noted in [1], a survey on quality assessment for Linked Data, *dataset completeness* commonly refers to a *degree* to which the "information *required* to satisfy some given criteria or query" are present in the considered dataset. Seen as a set of information objects, an ontology – or, more generally, a knowledge base (KB) –

[1] Corresponding Author, Corresponding author, Book Department, IOS Press, Nieuwe Hemweg 6B, 1013 BG Amsterdam, The Netherlands; E-mail: bookproduction@iospress.nl.

is a dataset (in [1] too). *KB objects* are either types or non-type objects. These last ones are either statements or individuals. A statement is a relation or a non-empty set of relations. In the terminology associated to the RDF(S) model [2], relations are binary, often loosely referred to as "properties" and more precisely as "property instances".

Extrinsic (dataset) completeness. In surveys referring to the *completeness of an ontology or KB*, e.g. [1,3,4], this notion is associated to the comparison of the KB to (real or idealized) reference KBs or to expected results when using such *other* KBs – hence, this article calls this notion "extrinsic model based completeness". E.g., *completeness oracles* [5], i.e. rules or queries *estimating* the information missing in the KB for answering a given query correctly, refer to an idealized KB. [3] distinguishes "gold standard-based", "corpus-based" and "task-based" approaches. [1] refers to schema/property/population completeness, and almost all metrics it gives for them are about *relatively* how many objects from a reference dataset are represented in the evaluated dataset.

Intrinsic completeness and, more generally, connectedness. This article gives a generic model to specify measures for the "semantic (intrinsic) completeness" notion(s). Each of these measures is a metric about *relatively how many* objects in a given set satisfy a semantic specification, i.e. one that specifies the semantic relations that each evaluated object should be source or destination of. *More generally,* "semantic *connectedness*" covers such a completeness *as well as* a simple count of objects satisfying the semantic specification. Thus, this notion is not similar to the "ontology completeness" of [6] where four "completeness theorems" define whether a KB is complete wrt. a specification stated in first-order logic. In [4], based on the descriptions and examples it gives, what its authors call "intrinsic completeness" covers semantic completeness ("ontology compliance" for these authors) and seemingly also completeness oracles.

Purposes. Unlike extrinsic model based completeness, semantic completeness is adapted for evaluating the degree to which a given set of objects complies with *ontology design recommendations (ODRs)*, such as particular ontology patterns [7], best practices [8] or methodologies (e.g. Methontology, Diligent, NeOn and Moddals). Such an evaluation eases the task of selecting or creating better KBs for knowledge sharing, retrieval, comparison or inference purposes.

Need for a generic specification model. Many KB evaluation measures can be viewed as connectedness measures for particular relation types. Many checks performed by ontology checking tools – e.g. Oops! [9] and OntoSeer [10] – also evaluate particular cases of semantic connectedness. However, it seems that no previous research has provided a generic way to specify semantic connectedness measures and thence enable their categorization and application-dependent generalizations (executable non-predefined ones), whichever the evaluated kinds of relations – and hence, whichever the domain, which explains why no domain, dataset or kind of datasets is referred to in this article. It is then also difficult to realize that *many existing KB evaluation criteria or methods are particular cases* of a same generic one.

Related research questions. In addition to this genericity issue, some research questions – which are related and apparently original – are then: i) how to define semantic connectedness, more precisely than above, and not only in a generic way but also one that is automatically checkable, ii) how to extend ODRs and represent knowledge for supporting an automatic checking of the use of particular relations while still allowing knowledge providers to sometimes disagree with such a use (this for example rules out checking that a particular relation is asserted whenever its signature

allows such an assertion), and iii) how to specify semantic connectednesses for the increase or maximization of the entering of particular relations by knowledge providers and then of inferences from these relations (e.g., especially useful relations such as subtype and exclusion relations, or type relations to useful meta-classes such as those of the OntoClean methodology [11]). These questions are important when the representations have to be precise or reusable, as is for example the case with foundational ontologies, or ontologies for knowledge sharing purposes.

Plan. To answer these research questions – the answers being the main contributions of this article – Section 2 introduces the proposed generic model by describing the parameters of *C**, one possible polymorphic function theoretically usable for checking any of the semantic connectednesses described in this article. For practical uses, CN, CN– and C%, three restrictions of C*, are also introduced. Any particular set of parameters of C* specifies one particular connectedness check. For genericity purposes too, the notion of aboutness is introduced as a generalization of the notions of meta-statements, modalities and negations. Then, via examples, Section 3 illustrates the flexibility and genericity of the approach. A simple user interface showing some interesting kinds of parameters is also provided. For explanatory and motivational purposes, all this is provided before Section 4 more precisely defines C%, CN and CN–; however, Section 4 can also be read first. For particular parameters, formal definitions are given; from them, definitions for other parameters can be derived.

2. Genericity: Generic Functions (C*, CN, CN-, C%) , "Aboutness" Relations

C* and the kinds of parameters it requires. Theoretically, a complex enough function – here named C* – could implement all elsewhere implemented *semantic connectedness checks*, although its code might have to be often updated to handle new features. Since the basic kinds of data used by C* can be typed and aggregated in different ways, C* could have different kinds of parameters, i.e. different signatures, even using "named parameters" (alias "keyword arguments", as opposed to positional parameters). In this article, to ease the readability and understanding of the proposed handy restrictions of C*, positional parameters are used and the selected untyped signature of C* is "*(objSelection1, objSelection2, constraints, metric, nonCoreParameters)*". Before describing these parameters, it should be noted that, to be generic, C* has to be polymorphic: for each parameter, C* should accept different kinds of objects, e.g. a set of objects, or a set of criteria to retrieve such objects, or a function to make that retrieval.

- Together, *objSelection1* and *objSelection2* specify the set of objects and/or relations to be checked, i.e. i) the set of objects from which particular relations are to be checked, and/or ii) the set of particular relations to check, and possibly iii) the set of objects that the destinations of the checked relations may be. In the use examples for the handy restrictions of C* given below, i) o*bjSelection1* is a set, typically "`{every rdfs:Class}`" to mean that the set of objects to be checked is the set of classes in the current KB, ii) *objSelection2* is a set, e.g. "`{rdfs:subClassOf, owl:equivalentClass}`" to mean that subClassOf relations and equivalentClass relations should be checked, and iii) the set of possible destination objects for these relations is not specified: by default, any destination is allowed.

- The 3rd parameter specifies *constraints* that the "objects and/or relations selected via the first two parameters" should satisfy. *E.g.:* for *each* selected object and relation type, there should be *at least one* relation of that type from this object.
- The 4th parameter specifies the metric to be used for reporting how many – or relatively how many – of the "objects and/or relations selected via the first two parameters" comply with the "constraints specified via the 3rd parameter". Examples of metrics and metric names are: i) "N_obj", the number of compliant selected source objects, ii) "N_rel", the number of compliant relations from&to the selected objects, iii) "L_obj–", the list of non-compliant source objects, iv) "%_obj", the ratio of *N_obj* to the number of selected objects, and v) "%_rel", the ratio of *N_rel* to the number of selected relations. More complex metrics can be used, such as those of the kinds described by [12] (e.g. "precision and recall" based ones) and [13] (e.g. "Tree Balance" and "Concept Connectivity").
- The 5th parameter specifies objects that are not essential to the specification of a semantic connectedness, e.g. parameters about how to store or display results and error messages.
- To sum up, the above distinctions (<selections, constraints, metric>) and associated parameters seem to support the dispatching of the basic kinds of data required by C* into a *complete* set of *exclusive* categories for these basic kinds of data, i.e., into a partition for them. Thus, all the data can be dispatched without ambiguities about where to dispatch them. The above parameters can also be seen as an handy way to describe parts of the model used in this article (the more common way to describe a model is to define tuples of objects).

CN, CN– and C% as handy restrictions of C*. CN, CN– and C% only have the first three parameters of C*. Using CN is like using C* with the *N_obj* metric as 4th parameter. CN– is like C* with the *L_obj–* metric (it is more useful during KB building than when comparing KBs). C% is like C* with the *%_obj* metric. From now on, unless otherwise specified, i) descriptions about C* are *also* about CN, CN– and C%, and ii) the word "specification" refers to a semantic connectedness specification for a KB, typically via C% since it allows the checking of a 100% compliance. Section 4 further defines these functions. Before that, Section 3 illustrates some uses of C% and CN–.

Conventions. In this article, a *type* is either a *class* or a *relation type* (alias, "*property*" for binary relations). Identifiers for relation types have a lowercase initial while other object identifiers have an uppercase initial. "OWL" refers to OWL-2 [14], the W3C ontology for an extension of the SROIQ description logic. Some of its types are used below for illustration purposes, mainly in example specifications. "RDFS" refers to RDFS 1.1 [2]. OWL types are prefixed by "`owl:`", and RDFS types by "`rdfs:`". The other types used in this article are declared or defined in an ontology named "Sub" [15] (a good part of it is about *sub*types, *sub*parts and similar relations; this ontology has over 200 types). E.g., Sub includes `sub:owl2_implication`, the most general type of implication that an OWL inference engine can exploit or implement. In accordance with the paragraph about genericity below, "`=>`" and symbols derived from it are not prefixed in the examples and definitions. Two statements or two non-empty types are in *exclusion* if they *cannot* have a shared specialization or instance, i.e., if having one is considered an error. E.g., `owl:disjointWith` is a type of exclusion relation between two classes.

Positive statements, negations, contextualizations and aboutness relations. In this article, i) a *statement* is a relation or a non-empty set of relations, ii) a *meta-statement*

is a statement that is – *or can be translated into* – a relation stating things *about* a(n inner) statement (generally, the source of the relation), and iii) this relation is called an *aboutness* relation. A negated statement can be seen as – or represented via, or converted into – a statement using a "not" relation expressing a "not" operator. A meta-statement that modifies the truth status of a statement – e.g., via a relation expressing a negation, a modality, a fuzzy logic coefficient or that the inner statement is true only at a particular time or place or according to a particular person – is a contextualizing statement (alias, *context*) for its inner statement, the contextualized statement. Thus, a relation that changes the truth status of a statement is a contextualizing relation, e.g. (in this article) a "not" or "necessarily not" relation. This article assumes that, if a contextualizing statement does not directly use a contextualizing relation, this statement is automatically converted (before calling C* or when the KB is loaded) into one that uses a contextualizing binary relation. A statement is either positive (i.e. without meta-statement, or with a meta-statement that simply annotates it instead of contextualizing it), negative (alias, negated), or contextualized but not negated. In this article, it is assumed that all contextualizing binary relation types used in the KB are defined as subtypes of `sub:contextualization`. All these previous notions enable a *generalization* of the specification and checking of what the introduction called ODRs (ontology design recommendations): *from* "the assertion of particular relations between particular objects" (the *classic* kinds of specifications) *to* "the assertion, negation or other contextualization of these relations". This simple change solves the three "related research questions" listed in the introduction. Indeed:

- Knowledge providers can now always state something *about* an advocated relation – e.g. that it is true, false or true only in a particular case – and thus comply with the above cited extended kind of specification, whereas it is not always relevant or possible to assert a particular (*classically*) advocated relation. Then, a provided *statement about an advocated relation* can be automatically checked. Conversely, automatically checking *classic* specifications (thus, the assertion of particular relations) is problematic since any non-existence or contextualization of an advocated relation has to be assumed to be a mistake whereas this may have been for conceptual reasons: in particular cases, the advocated relation was irrelevant or needed to be contextualized.
- Having *knowledge providers systematically state (something about) particular relations* increases or maximizes their entering and, as especially illustrated by the fifth paragraph of Section 3, inferences from them.

Genericity wrt. inference engines. For genericity purposes, the previous paragraph uses the notion of *contextualization*. This does not mean that the approach *requires* contexts or a second-order logic since in many KBs contexts are not needed or not used, or partial (or *ad hoc*) representations of them are used, e.g. i) exclusion relations can replace some uses of the "not" operator or the "necessarily not" of athletic modal logic, and ii) contexts in general may for example be expressed via the "Context Slices" design pattern [16] or via statement reification (e.g., as in RDF) plus types for particular kinds of contexts. Instead of calling an *external* logic-based inference engine, some implementations of connectedness checking functions may use simple graph matching techniques – which may take into account partial (or *ad hoc*) representations of contexts – when searching objects and relations that *match* specifications given via the parameters. This is how these checking functions are implemented in WebKB-2 [17], thus how the examples were validated and the evaluations performed. A *companion*

Web article [18] also proposes SPARQL+OWL queries as implementations of these functions for particular kinds of parameters (e.g., those of all the examples below related to the checking of classes), and these queries have been validated via Corese [19], a SPARQL and OWL inference engine. The interface shown in Figure 1 is able to call both tools. For genericity purposes, the approach presented in this article is purposefully not related to a particular logic, knowledge representation language (KRL), inference engine or strategy. To that end, this article uses the expression "*the used inference engine*" (be it internal or external to the function) and the "=>" symbol. It refers to the implication operator of the KRL exploited by *the used inference engine*. Although Section 4 gives a second-order logic formula (that uses "=>"), this one can be downgraded – e.g., instantiated wrt. each of the types in the KB – to match the used KRL. Then, the logical properties of the checking function and approach are derived from those of the used "=>" and engine:

- If the used logic is "syntactically or semantically valid" (informally, both cases imply that if the premises of an implication is true, its conclusion cannot be false), then the approach is "syntactically or semantically valid".
- If the formal system composed of the KB and the used logic is "semantically complete" (i.e., if all the tautologies of this system are provable), then C* gives all the results it should. If that system is not semantically complete, some compliant or non-compliant objects may not be retrieved.
- The computational properties of the function depends on i) which operation of the inference engine it calls – and how repetitively it is called – to perform the above cited search and matching, and ii) the computational properties of this operation: they depend on the entailment regime [20] of the engine as well as the normalization [21] and complexity of the exploited KRs (in OWL [14], this relates to the notion of "profile" [22], e.g., OWL-2 EL is a fragment of OWL-2 [14] that has "polynomial time reasoning complexity"; [22] gives computational properties for each OWL-2 profile).
- To conclude, although the results of the function depend on the selected inference engine, it can be said that the approach itself is independent of a particular inference engine. This kind of genericity is an advantage and, at least in *this* article, there would be no point in restricting the approach to a particular logic.

Comparison with constraints. In a KB, constraints enforce the existence of particular relations. Hence, using them is similar to using C% and considering that a result inferior to 100% is a problem. However, constraint languages – e.g. SHACL [23] – generally do not allow the use of contexts in specifications, let alone specifications about "positive *or* contextualized relations of particular types". If the goal is to compare KBs, using constraints is also not an easy method.

3. Illustrations and Experimental Validations of the Genericity of the Approach

Notation. For concision and clarity purposes, i) the notation used below is a recent extension of FL [17], and ii) some parameters are *predefined* strings with particular meanings for the functions, e.g. `"every object to some object"`.

"Coverage of a class" in the sense used in [24]. In [24] (unlike in [25]), the "coverage" of a class in a KB is the ratio of i) the number of instances of this class, to ii) the number of instances (in the KB). For a class identified by `Cl`, such a coverage could be measured via `C%( {every owl:Thing}, {rdf:type -> Cl}, {"every object to some object", "no negated or other contextualized relation is counted"} )`: due to the cardinalities `"every object to some object"` in the 3rd parameter, this call returns 100% if and only if *every* object of type `owl:Thing` (e.g. a class if there are meta-classes) is source of *some* (i.e., at least one) `rdf:type` relation to `Cl`. This relation must be positive due to the `"no negated or other contextualized relation is counted"`. This restriction could also be written as `[any sub:Statement --sub:counted-contextualizing-relation-types--> {}]` which can be read: "any statement has for `sub:counted-contextualizing-relation-types` an empty set" (thus, no contextualized statement can comply with a specification).

Existential completeness wrt. particular relations: existence of relations of particular types from every selected object to some object. The expression "existential completeness" refers to the use of the cardinalities `"every object to some object"` in the 3rd parameter (as in the previous paragraph) but *also* emphasizes that the goal or ideal when using that specification is to have a KB fully complying with it (unlike in the previous paragraph). An an example, `C%( {every rdfs:Class}, {rdfs:label, rdfs:comment, rdfs:subClassOf}, {"every object to some object", "no negated or other contextualized relation is counted"} )` returns the percentage of classes in the checked KB that are source of *at least* three (asserted or inferable) *positive* relations (to *some* objects in the KB): an `rdfs:label` relation, an `rdfs:comment` relation and an `rdfs:subClassOf` relation. The "*at least* three" is because in the used notation "{…}" sets are AND-sets (by default, but OR-sets and NOT-sets can also be used; all these kinds of sets can also be combined).

Universal completeness wrt. particular relations (here, subclassOf and equivalentClass relations): existence of relations of particular types from every member of the set of source objects to every member of the set of destination objects. Analogously to the previous example, `C%( {every rdfs:Class}, {rdfs:subClassOf}, {"every object to every named object", [any sub:Statement --sub:counted-contextualizing-relation-types--> {sub:negation}] } )` returns 100% if every class in the KB is connected (by an asserted relation or one inferable by the used engine) to *every* named object in the KB via positive `rdfs:subClassOf` relations *or negated* ones (due to the last part in the 3rd parameter). A "named" object is one that is identified (e.g., a non-anonymous type) or is source of an `rdfs:label` relation. When the engine knows that the destination objects can only be classes (e.g., via a specification in the parameters or since no statement can be a class), the above call returns 100% if *every pair of (named) classes* is connected by a positive or negative `rdfs:subClassOf` relation. Similarly, `C%( {every rdfs:Class}, {rdfs:subClassOf, owl:equivalentClass}, {"every object to every named object", [any sub:Statement --sub:counted-contextualizing-relation-types--> {sub:negation}] } )` gives the percentage of classes satisfying the following condition: between every evaluated pair of classes, there should now be a(n asserted or inferable) positive or negative `rdfs:subClassOf` relation *and* a positive or negative `owl:equivalentClass` relation. Thanks to this last one, even though `rdfs:subClassOf` relations are not strict (their type is not disjoint with `owl:equivalentClass`), the use of *strict subclassOf* relations is actually evaluated. I.e., if `rdfs:subClassOf` relations are used between two classes, there must also be a relation

indicating whether these classes are equivalent. This detects unintended `rdfs:subClassOf` cycles between classes.

Universal completeness wrt. generalization, equivalence and exclusion relations between types. A specification even more useful than the previous one is: `C%( {every rdfs:Class}, {rdfs:subClassOf, owl:equivalentClass, owl:disjointWith}, {"every object to every named object"} )`. It returns 100% if *every pair of classes* is connected by a relation for each the three specified types and this relation may be positive, negative or otherwise contextualized since there is now no restriction on contexts in the 3rd parameter. A KB complying with such a specification (with any context allowed or, as in the previous paragraph, only negations) has at least two advantages. First, the "closed-world assumption" (i.e., any statement not represented in the KB is assumed to be false) and the "unique name assumption" (i.e., different identifiers are assumed to refer to different things) do not lead to any more inferences regarding relations of the three above cited types: at least for these relations, these assumptions are not needed. Second, in such a KB, querying types or their instances based on their exclusions leads to interesting results. Creating such a KB is not cumbersome when i) the inference engine can deduce that the subtypes of a type are disjoint with its disjoint types and then also cannot subtype – nor be equivalent to – these disjoint types, and ii) whenever appropriate, knowledge providers use `owl:disjointUnionOf` (or else `owl:unionOf`) objects and relations of types such as `sub:sC` (defined in Sub via FL – and also via OWL+SparqlUpdate – as a strict subclassOf relation type such that the source subclass is neither exclusive to, subtype nor supertype of any of its siblings). Based on the above examples, *universal completeness* – i.e., the use of the "every to *every*" kinds of cardinalities in the 3rd parameter – seems more useful than *existential completeness*. Hence, from now on, `"every object to every named object"` is a default specification in the 3rd parameter. Thus, `C%( {every sub:Type}, {sub:supertype, sub:equivalent_type, sub:disjoint_type} )` generalizes the previous specification to all types.

Universal completeness wrt. implication, equivalence and exclusion between statements. With "=>!" (alias, "=>¬") being the type of "exclusion between two statements" derived from "=>", `C%( {every sub:Named_statement}, {=>, <=>, =>!} )` is analogue to the previous specification but applies to named statements, i.e., those that have been reified and named. If a KB complies with that specification, all named statements are organized via positive or contextualized "=>" relations (that are manually set or that can be deduced by the inference engine) into (or wrt. objects in) a "=>" hierarchy where objects are also connected by equivalence and exclusion relations, whenever possible. These relations can be deduced by the used inference engine *if* the types of the KB comply with the last specification of the previous paragraph and *if* the used inference engine can fully exploit the content of the statements (this implies that this content is fully formal, hence not partially represented via strings, and that the used logic is decidable). This hierarchy may be useful for performance or explanatory purposes. The specification may also be extended and exploited by the editing protocol of a shared KB for enabling its users to cooperatively update it while keeping it free of inconsistencies or redundancies, without restricting what the users can enter nor forcing them to agree on terminology or beliefs [18].

Completeness of the destinations of subtype relations: existence of at least one "complete set of destinations". A call to `C%( {every rdfs:Class}, {sub:subClass}, {[* → 1..* complete{…}]} )` returns 100% if *every class that has at least one subclass* has at least one set of subclasses that is *complete* (in the usual sense for such a set: each

instance of the class is instance of at least one of the subclasses; `owl:unionOf` relations can be used for representing such a set). Similarly, `C%( {every rdfs:Class}, {sub:subClass}, {[* → 1..* partition{…}]} )` returns 100% if *every class that has at least one subclass* has at least one "subclass partition", i.e. a set of exclusive subclasses that is *complete* (in the same sense as previously; `owl:disjointUnionOf` relations can be used for representing such a set).

Specification of most of the checks made by Oops!. The ontology checking tool Oops! [9] can semi-automatically check 33 "common pitfalls", among which 9 inconsistency problems and 14 "missing value" problems that can be seen as semantic connectedness specifications. E.g., "P01: Creating unconnected ontology elements" problems can be detected via `CN-( {every owl:Thing}, {sub:relation} )` while "P11: Missing domain or range in properties" problems can be detected via `CN-( {every rdf:Property}, {rdfs:domain, rdfs:range} )`. The inconsistency problem "P06: Including cycles in a class hierarchy" can be detected via `CN-( {every rdfs:Class}, {rdfs:subClassOf, owl:equivalentClass} )`. The missing value problem "P13: Inverse relationships not explicitly declared" can be detected via `CN-( {every rdf:Property}, {owl:inverseOf} )`. [26] advocates some checks related to some of OOps!. One way to perform many of them is to call `CN-( {every rdfs:Class}, {rdfs:subClassOf, owl:equivalentClass} )` and `CN-({every rdf:Property}, {rdfs:domain, rdfs:range} )`.

Evaluation of a well known foundational ontology. To illustrate one experimental implementation and validation of this approach, DOLCE+DnS Ultralite (DUL) [27] – one of the most used foundational ontologies and one represented in RDF+OWL – has been checked via `C%( {every rdfs:Class}, {rdfs:subClassOf, owl:equivalentClass, owl:disjointWith} )`. More precisely, an automatic check was made on an extension of this ontology ("DUL 3.32 + D0 1.2", from the same author; version of April 14th, 2019) but it is still named DUL below. For understandability and analysis purposes, [28] gives an FL-based and modularized very slight extension of this ontology. The first result was 0%: no DUL class has a positive/contextualized asserted/inferable relation to *every* class for *each* of the above listed types. Partial reasons for this are: i) DUL uses `rdfs:subClassOf` instead of a strict subclassOf relation, and ii) it has few `owl:disjointWith` relations. However, only a few exclusion relations had to be added to DUL for the following assumption to be true: no class is equivalent to any other class and no class has other potential supertypes, subtypes and exclusions than those explicitly represented. Then, for making this explicit – i.e, for this assumption to be unneeded – the `rdfs:subClassOf` relations were replaced by more precise ones (typically of the above cited `sub:sC` type); this made the modified version of DUL automatically checkable via the above cited C% call and then the result was 100%. Given the names and comments associated to DUL classes, the relations added for making the above assumption true seemed warranted. For DUL, with some weaker assumptions, the maximum result was 11.9% (more precisely 10/84). The interested readers can find more details in [18]. The organization of relation types has been similarly checked via `C%( {every rdf:Property}, {rdfs:subPropertyOf, owl:equivalentProperty, owl:propertyDisjointWith} )`. The results were also 0% when no assumption was made and 100% (more precisely, 112/112) when the above cited one was made. However, to make this assumption true, a lot of seemingly warranted exclusion relations and non-exclusion relations had to be added between the types. Some other top-level ontologies were similarly checked and the results were similar. This is not surprising: nowadays, even in foundational ontologies, it is rare that subtype partitions or sets of exclusive subtypes are used *whenever*

possible (and, it is even rarer that non-exclusion relations are set for making explicit to the inference engine that some types *cannot be* related by exclusion relations). Nevertheless, as earlier noted, in the general case, adding such relations is easy and support inferences that may prove valuable for some applications (this does not mean that, for *most* current applications, such relations would lead to better results or a better performance; this would also be irrelevant for knowledge modelling&sharing purposes).

Comparison to the measure named "coverage" in [25]. In [25], the "coverage of a class within a dataset" is with respect to the "properties that belong to the class". For each of these properties (binary relations from the class), this coverage is (very informally) the ratio of i) the number of occurrences of this property in (all) the instances of this class, to ii) the product of "the number of properties in this class" and "the number of instances of this class (in the evaluated dataset)". This *coverage* was designed to return 100% when *all* instances of a class have *all* the "properties that belong to the class" (to use the terminology of [25], one more often associated to some frame-based KRLs than to more expressive KRLs). To represent and generalize this last expression, C* and its derived functions can exploit the special variable (or keyword) "`$each_applicable_relation`" in their 2nd parameter. This variable specifies that "*each* relation type (declared in the KB or the KBs it imports) *which can be used* (e.g., given its definition or signature) *should be used whenever possible, directly or via a subtype*". E.g., for a class identified by `Cl`, a call to `C%( {every Cl}, {$each_applicable_relation}, {"every object to some object"} )` would return the ratio of i) the number of instances of `Cl` that have at least one relation of each of the possible types, to ii) the number of instances of `Cl`. Thus, 100% would be returned when *all* instances of `Cl` have (at least one instance of each of) *all* the relations they can have. This is not the coverage of [25] but has a similar intent and is compatible with expressive KRLs. To compare KBs, [25] advocates the use of the "*coherence* of a class within a dataset"; it is the sum of a weighted average of the coverages of the classes, thus not a ratio between comparable quantities and not a particularly intuitive measure. With C%, comparing KBs based on similar coverages of their classes could instead be done by calling `C%( {every rdfs:Class}, {$each_applicable_relation}, {"every object to some object"} )` for each KB and then comparing the results.

A simple user interface showing some interesting kinds of parameters for semantic connectedness checks. Figure 1 shows a user interface that i) helps people build parameters for some functions like C%, ii) generates a query (or, in some cases, a SHACL constraint), and iii) calls a KB server (endpoint) with the query or constraint. This server displays the results of the execution of the query or of the adding of the constraint. For functions, this interface was tested with WebKB-2 [17]; for SPARQL+OWL or SHACL, a local Corese server [19] was used. Each of the points below comments on one menu. These points are only meant to give an overview of interesting options and general ideas about what can be achieved. The interested readers can find more details in [18]. In these menus, the indentations represent specializations.

- The "Level" menu on the top right corner selects options in the four selection menus about the parameters: those in the middle of Figure 1. This figure shows the selection of the options for the level named "Good (/ Owl+Sub)". OWL-RL or OWL-QL, along with some Sub relations defined with these OWL profiles, are sufficient for *class definitions* to *fully comply with the options of this level* and to check this compliance (but this is often not the case for *relation type definitions*, statements and individuals).

- The “From such objects” menu enables the selection of some particular types and quantifiers for the 1st parameter of C* (thus, this menu is in the 1st column), hence for the objects in the sources of relations selected to be checked.
- The “To such objects” menu is about which destination objects may be used when exploiting the “to” part of specified cardinalities (e.g. "every ... *to* ..."). Thus, although in Figure 1 this menu had to be placed in the 1st column, it is related to the 3rd parameter of C*. This menu is mainly only useful for universal completeness ("every ... to *every* ..."). With the “accessible objects” option, the chains of relations that start from the evaluated source objects are followed across KBs. At least in the RDF world, this means *dereferencing* the URIs of each reached object to find KBs that may specify additional relations on this object. This option can be implemented via the SPARQL of Corese [19] but not SPARQL 1.1. With the “in the KB” option, the destination objects may only be those in the evaluated KB and the KBs it imports. The “last added object” option is for checking any object addition to a KB before fully accepting this object. This is for example useful when a KB is cooperatively built or loaded via sequence of assertions or, more generally, of commands.
- The “Via ... relations of these types” menu shows types (of possibly contextualized or deducible relations) that are particularly interesting to check. The “`sub:`” prefix is not displayed. The type “`==>`” generalizes `sub:supertype` and the type for *implications (“=>”) that are also generalizations*. The types “`==>!`” and “`<==>`” are the exclusion and equivalence types related to “`==>`” such that using `C%({every owl:Thing}, {==>, <==>, ==>!})` is equivalent to using both `C%({every sub:Type}, {sub:supertype, sub:equivalent_type, sub:disjoint_type})` and `C%( {every sub:Statement_for_inferences}, {=>, <=>, =>!} )`. A complying KB maximizes inferences wrt. “`==>`” and “`==>!`”. The type “`==>-element`” generalizes i) “`=>-element`”, the type of relations *from* a statement *to* a non-statement object used in a “clause” (an AND part in a conjunctive normal form) of what is implied by this statement in its KB, and ii) `sub:NC-definition_element`, the type of relations *from* a type *to* a non-statement object (hence a type or an individual) directly or indirectly used in a *definition by necessary conditions* of that type (this relation is transitive, hence the “directly or indirectly used”). Section 5 of [18] defines both “`==>-element`” and the related “`==>-element_exclusion`” in *second-order logic* as well as *in OWL for class definitions written in OWL*, and shows how `C%( {every sub:Type}, {sub:definition-element, <==>, sub:definition-element_exclusion} )` enables the detection of implicit redundancies in a KB.
- Most of the options shown in the “Other checked constraints” menu are related to the 3rd parameter of C*. Section 2.4.5 of [18] shows how a universal completeness specification with the “*minimal differentia* (between any two selected objects)” option is a way to define, extend and implement the “Differential Semantics” methodology of [29], itself an extension of the *genus & differentia* design pattern. E.g., [18] shows that it can be implemented via CN– and a SPARQL query. Similar options are proposed to obtain similar structures for the organization of “`==>`” relations and `sub:part` relations in a KB. Other options in this menu are about previously explained kinds of parameters (e.g., what is above called “cardinalities”).

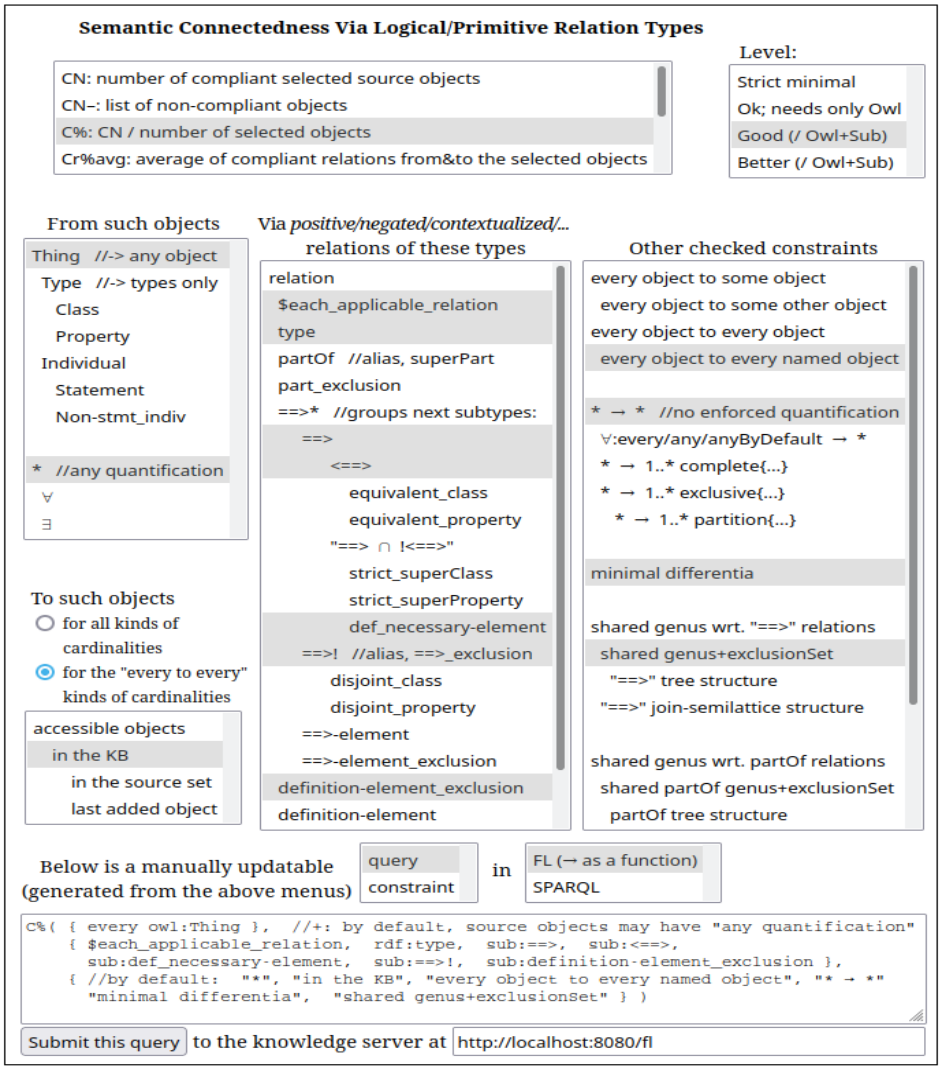

Figure 1. A simple interface for the evaluation of semantic connectedness.

4. More Precise Definitions of C*, CN, CN–, C% and CNΔ

Informal definition of C* with default parameters: "every object to every named object" cardinalities with destination objects in the KB; no restriction on the destination objects nor on the types of contextualizing relations. With its default parameters, C* checks that, in the tested KB, for *every* selected relation type, there is at least one relation of this type – or, more precisely, at least one statement *about* a relation of this type – *from every object* of the selected set *to every (other or not) named object* of the KB. In this check, "at least one statement *about* a relation" uses the previously introduced notion of *aboutness* and means that i) there exists a statement asserting whether the relation exists, does not exist, cannot exist or exists under certain conditions, and ii) this statement is

asserted, or is hard-coded in the inference engine used for the evaluation, or can be inferred by this engine without using the closed-world assumption or the unique name assumption (since the goal if for knowledge providers to make the necessary information explicit).

More formal definition of CN for its above listed default values, with a set of binary relation types as the 2nd parameter (this restriction is for clarity purposes; from this definition, other ones without this restriction or for other parameters can be derived; a (second-order) logic based notation is here used for clarity and concision purposes but, as earlier noted, CN implementations do not need to exploit a second-order logic; they can also be fully function based; [18], the companion article, also proposes SPARQL+OWL queries for particular cases). With `Kb` being a KB or portion of KB, and `NamedObjs` being the set of named objects in `Kb`, for any type `OT` and any set of binary relation types that is identified as `RTs`, calling `CN({every OT}, RTs, {})` returns the number of objects `O1` satisfying the next formula and its two associated points.

- *Formula 1*: `∀ rt ∈ RTs, O1 ∈ Kb, O2 ∈ NamedObjs ∃ rto`
  `OT(O1) ∧ rdfs:subPropertyOf(rto,rt) ∧`
  `( (Kb => rto(O1,O2)) ∨ (Kb => ¬ rto(O1,O2)) ∨`
  `(Kb => (∃c sub:Contextualization(c) ∧ sub:contextualization(rto(O1,O2),c)`
  `) ) ).`
- As previously justified, "`=>`" refers to the implication exploited by *the used inference engine* (Section 2), without feature selection leading to the use of the closed-world assumption or the unique name assumption (Section 3). Since `rdfs:subPropertyOf(rt,rt)` is reflexive, `rto` may be identical to `rt`. "¬" ("`!`") is the classic negation operator.
- The type `sub:Contextualization` is the type for all contextualizing conditions or values, e.g. the truth status `sub:False`, while `sub:contextualization` is the relation type for all binary relations from a statement to a contextualizing condition or value. It is assumed that, when needed, statements in the KB can be – or have been – converted by the used inference engine into statements that use instances of those types. The "`(Kb => ¬ rto(O1,O2))`" part of Formula 1 is only there in case negations have not explicitly been defined as contextualizations: this part is redundant if the KB includes the following assertion which makes negation a particular meta-statement.
  `∀ r,x,y ¬ r(x,y) <=> sub:contextualization( r(x,y), sub:False ).`

CN for the "every object to some object" cardinalities. To obtain the counterpart of Formula 1 for these other cardinalities, "`∀ rt ∈ RTs, O1 ∈ Kb, O2 ∈ NamedObjs`" is to be replaced (within Formula 1) by "`∀ rt ∈ RTs,O1 ∈ KB ∃ O2`". By adding "`∧(O1 != O2)`" at the end of the formula, the "every object to *some other* object" cardinalities are used.

Definition of C% and CN–. C% divides the result of CN by the number of evaluated objects, i.e. by the number of objects of type `OT` if the terms and conditions for Formula 1 are used. With these conditions, if Formula 1 is satisfied, C% returns 100%, while CN– returns the list of objects of type `OT` for which Formula 1 is false.

Restrictions on the counted contextualizing relation types. Section 3 illustrated the restrictions `[any sub:Statement --sub:counted-contextualizing-relation-types--> {}]` and `[any sub:Statement --sub:counted-contextualizing-relation-types--> {sub:negation}]`. Logically speaking, using any of these two constraints means dropping some arguments of the "or" expression in Formula 1: for the first constraint, the 2nd and 3rd arguments are dropped, i.e., only the argument "`(Kb => rto(O1,O2))`" remains; for the second constraint, only the 3rd argument of the "or" expression is dropped.

Specification of mandatory contextualizing relation types. Using `sub:`*`mandatory`*`-contextualizing-relation-types` instead of `sub:`*`counted`*`-contextualizing-relation-types`, one may specify the types of the contextualizing relations that are *mandatory* for the checked relations, instead of just *taken into account* in the ways previously described. Logically speaking, with `MRTs` referring to a set of mandatory binary relation types, this means replacing the "or" expression in Formula 1 by "`(Kb => (∀ mrt ∈ MRTs (∃c sub:Contextualization(c) ∧ mrt( rto(O1,O2), c ) ) ) )`". E.g., `[any sub:Statement --sub:mandatory-contextualizing-relation-types--> {sub:time}]` means that each of the checked relations should have temporal contextualizations. When these contextualizations are not explicitly represented via a meta-statement – i.e., when they are implicit (e.g. hard-coded in the inference engine) or represented in another way – these contextualizations should be inferred for the checking to work as expected.

CNΔ (relation usefulness). `CN({every sub:Named_statement}, {=>, <=>, =>!})` gives the number of named statements related by positive or contextualized relations of the three indicated types. This number may be seen as indicating the number of inferences (based on these types) between named statements in the KB. This number can be obtained right before and right after a relation is added to the KB – added *explicitly*, not *inferred*. Then, CNΔ – the difference between the two obtained numbers – is the number of additional inferences (of the cited kinds) that this added relation has led to.

5. Conclusion

Technical highlights. The intrinsic *ontology completeness* notions and, more generally the *semantic connectedness* notions, are the product of many sub-notions. This article showed i) some important sub-notions (Figure 1 is a synthesis and the beginning of a categorization), ii) that only few functions are needed for specifying and checking this product, and iii) that the approach it proposes also enables the automatic checking and generalization of some ODRs and related "KB quality measures". The provided examples and evaluation introduced some original and useful specifications which are rarely complied with (even by foundational ontologies) even though they would be easy to comply with. This article also showed that, in other research works, KB evaluation measures that can be categorized as semantic connectedness measures have far less parameters and do not exploit aboutness. Thus, they do not answer the research questions of this article. The metrics used by many of such measures are not simple ratios between comparable quantities (quantities of same nature): the proposed approach can use these metrics (via the 4th parameter of C*) or, as illustrated in Section 3 (*Comparison to the measure named "coverage" in [25]*), may provide alternatives.

Next steps. The companion article [18] shows the beginning of an ontology of criteria and relation types used in ODRs and KB quality measures. This ontology will be completed and exploited by the code and user interface made for implementing and validating the approach introduced in this article. This approach will be exploited to generalize the KB editing protocol of the KB sharing server WebKB-2 [17] and enable its users to adapt it. This approach will also be applied for checking relations automatically extracted from structural dependencies within software code (programs, library of software components, etc.), e.g. for checking the universal completeness of various types of partOf and generalization relations between software objects (functions, structures, variables, etc.).

References

[1] Zaveri A., Rula A., Maurino A., Pietrobon R., Lehmann J., Auer S. Quality assessment for linked data: A survey. Semantic Web 2016, 7(1): 63–93.

[2] w3c. RDF Schema 1.1. W3C Recommendation 25 February 2014, http://www.w3.org/TR/rdf-schema/

[3] Raad J., Cruz C. A survey on ontology evaluation methods. Proceedings of IC3K 2015, Lisbon, Portugal, p. 179–186.

[4] Wilson S. I., Goonetillake J. S., Ginige A., Walisadeera A. I. Towards a Usable Ontology: The Identification of Quality Characteristics for an Ontology-Driven Decision Support System. IEEE Access, 2022, 10:12889–12912.

[5] Galárraga, L., Hose, K. and Razniewski, S. (2017) Enabling completeness-aware querying in SPARQL. Proceedings of WebDB 2017, IL, USA, p. 19–22.

[6] Grüninger M., Fox M. S. Methodology for the Design and Evaluation of Ontologies. Proceedings of IJCAI 1995 Workshop on Basic Ontological Issues in Knowledge Sharing. Montreal, Canada.

[7] Dodds L., Davis I. Linked Data Patterns – A pattern catalogue for modelling, publishing, and consuming Linked Data. 2012 http://patterns.dataincubator.org/book/. http://ontologydesignpatterns.org

[8] w3c. Data on the Web Best Practices. W3C Recommendation 31/01/2017, http://www.w3.org/TR/dwbp/

[9] Poveda-Villalón M., Gómez-Pérez A., Suárez-Figueroa M. OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation. Int. J. Semantic Web Inf. Syst. 2014, 10 (2): 7–34, see also http://oops.linkeddata.es/catalogue.jsp

[10] Bhattacharyya P., Mutharaju R. OntoSeer--A Recommendation System to Improve the Quality of Ontologies. arXiv preprint arXiv:2202.02125, 2022.

[11] Guarino G., Welty C. An Overview of OntoClean., Handbook on Ontologies (DOI: 10.1007/978-3-540-92673-3_9), 2009, p. 201-220.

[12] Hartmann J., Spyns P., Giboin A., Maynard D., Cuel R., Suárez-Figueroa, M.C., Sure Y. D1.2.3 Methods for ontology evaluation. EU-IST Network of Excellence (NoE), 2005, IST-2004-507482 KWEB Deliverable D1.2.3 (WP 1.2).

[13] Ning H., Shihan D. Structure-Based Ontology Evaluation. Proceedings of ICEBE 2006, Shanghai, p. 132–137.

[14] w3c. OWL 2 Web Ontology Language – Structural Specification and Functional-Style Syntax (Second Edition). W3C Recommendation 11 December 2012, http://www.w3.org/TR/owl2-syntax

[15] Martin Ph. The Sub Ontology in Turtle. 2019, http://www.webkb.org/kb/it/o_KR/o_KB/o_upperOntology/dolce/d_dul_fl.html

[16] Welty C. Context Slices. 2010, http://ontologydesignpatterns.org/wiki/Submissions:Context_Slices

[17] Martin Ph. Towards a collaboratively-built knowledge base of&for scalable knowledge sharing and retrieval. HDR thesis (Habilitation to Direct Research; 240 pages), 2009, University of La Réunion.

[18] Martin Ph. Ontology Intrinsic Completeness and Connectedness. (companion Web article for the present article), 2022, http://www.webkb.org/kb/it/o_KR/p_kEvaluation/o-completeness/

[19] Corby O., Faron-Zucker, C. STTL: A SPARQL-based Transformation Language for RDF. Proceedings of WEBIST 2015, 11th Conference on Web Information Systems and Technologies, Lisbon, Portugal.

[20] w3c. SPARQL 1.1 Entailment Regimes. W3C Recommendation 21 March 2013, http://www.w3.org/TR/sparql11-entailment/

[21] Vrandečić D., Sure Y. How to design better ontology metrics. Proceedings of ESWC 2007, p. 311-325.

[22] w3c. OWL 2 Web Ontology Language Profiles (Second Edition). W3C Recommendation 11 December 2012, http://www.w3.org/TR/owl2-profiles/

[23] w3c. Shapes Constraint Language (SHACL). W3C Recommendation 20 July 2017, http://www.w3.org/TR/shacl/

[24] Karanth P., Mahesh K. Semantic Coverage Measures: Analytic Operators for Ontologists. Proceedings of KDIR 2016, 8th International Conference on Knowledge Discovery and Information Retrieval, Porto, Portugal, November 2016.

[25] Duan S., Kementsietsidis A., Srinivas K., Udrea O. Apples and Oranges: A Comparison of RDF Benchmarks and Real RDF Datasets. Proceedings of ACM SIGMOD 2011, p. 145-156.

[26] Gómez-Pérez A. Towards a framework to verify knowledge sharing technology. Expert Systems with applications, 1996, 11.4:519–529.

[27] Gangemi A. Ontology:DOLCE+DnS Ultralite. 2019, http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite

[28] Martin Ph. DOLCE+DnS Ultralite (DUL) in FL. 2021, http://www.webkb.org/kb/top/dolce/d_dul_fl.html

[29] Bachimont B., Isaac A., Troncy R. Semantic Commitment for Designing Ontologies: A Proposal. Proceedings of EKAW 2002, LNCS, volume 2473, p. 114–121, Springer Berlin, Siguenza, Spain.