Type-checking knowledge graphs

Iztok Savnik[1] and Kiyoshi Nitta[2]

[1] Faculty of mathematics, natural sciences and information technologies, University of Primorska, Slovenia
iztok.savnik@upr.si
[2] Yahoo Japan Corporation, Tokyo, Japan
knitta@yahoo-corp.jp

Abstract. We first present a formal view of a knowledge graph. On this basis, the type-checking rules are developed to define correct typing relationships among the triples of a knowledge graph. We discuss the algorithms for verifying the typing relationships against the given knowledge graph. Finally, we present the experimental results of type-checking the Yago4 knowledge graph.

Keywords: RDF stores · graph databases · knowledge graphs · database statistics · statistics of graph databases.

Table of Contents

1 Introduction

This is intro... [1].

2 Formal framework

This section describes the formal view of knowledge graphs

3 Typing knowledge graphs

3.1 Typing literals

3.2 Typing identifiers

– The following is view from the formalization.

The set I includes individual identifiers I_i, class identifiers I_c and predicate identifiers I_p. Let $i_1, i_2 \in I$. The relationship preceeds $\preceq$ on the set I is defined as follows. If the identifier i_1 is more specific than or equal to i_2 with respect to the conceptual schema of a knowledge graph, then $i_1 \preceq i_2$.

The relationship $\preceq$ defines a partial ordering of the identifiers from I that we denote $(I, \preceq)$. As described in the section on formalization, the class identifiers I_c stand for the types of individual identifiers I_i. Hence, the partial ordering $(I, \preceq)$ is defined by means of the relationships rdf:type, rdfs:SubClassOf and rdfs:subPropertyOf. In this way, we obtain also the isomorphical poset defined on the interpretations of individual types (classes) using the subsumption relationship $\subseteq$.
– We now state the above in the realm of the sub-typing relationship.

Stored sub-typing of identifiers.

– Partial ordering defined with triples in a database.
– The relationships that poset covers are rdf:type, rdfs:subClassOf and rdfs:subPropertyOf.
– All identifiers included in the relalationship $\preceq_1$.
– This allows us to separate and also address separately the ssg and subtyping relationship.
– The relation $\preceq_1$ includes solely the stored *relationships among identifiers.*
– The relation $\preceq$ is the relation $\preceq_1$ extended with the reflexivity and transitivity.

– Oportunity to introduce "mixed" objects including ground and schema components.

Reflecting the one-step relationship rdf:type in $(\mathcal{I}, \preceq)$.

$$\frac{I_1 \in \mathcal{I}_i \quad I_2 \in \mathcal{I}_c \quad (I_1, \text{rdf:type}, I_2) \in \mathcal{D}}{I_1 \preceq_1 I_2} \tag{1}$$

Including the one-step relationship rdfs:subClassOf in $(\mathcal{I}, \preceq)$.

$$\frac{I_1, I_2 \in \mathcal{I}_c \quad (I_1, \text{rdfs:subClassOf}, I_2) \in \mathcal{D}}{I_1 \preceq_1 I_2} \tag{2}$$

Including the one-step relationship rdfs:subPropertyOf in $(\mathcal{I}, \preceq)$.

$$\frac{I_1, I_2 \in \mathcal{I}_p \quad (I_1, \text{rdfs:subPropertyOf}, I_2) \in \mathcal{D}}{I_1 \preceq_1 I_2} \tag{3}$$

– Show that all identifiers are included.

Subtyping identifiers.

– Relate everything with subsumption poset.

Generalizing one-step relationship $\preceq_1$ to the relationship $\preceq$ in $(I, \preceq)$. $\preceq_1$ is a basis og $\preceq$.

$$\frac{I_1, I_2 \in \mathcal{I} \quad I_1 \preceq_1 I_2}{I_1 \preceq I_2} \tag{4}$$

Subtyping is reflexive.

$$\frac{S \in \mathcal{I}}{S \preceq S} \tag{5}$$

The subtype relationship is transitive. We require that the symbols S, U and T are identifiers. Note that S can be an individual identifier while U and T have to represent classes.

$$\frac{S, U, T \in \mathcal{I} \quad S \preceq U \quad U \preceq T}{S \preceq T} \tag{6}$$

Types include a special type $\top$ that represents the most general type in the ontology. Every type is more specific than the top type $\top$.

$$S \preceq \top \tag{7}$$

Typing of identifiers. A base type of an individual identifier I is a type related to I by the relationship $\preceq_1$. Derivation of base types of I is defined using the following rule.

$$\frac{I \in \mathcal{I}_i \quad C \in \mathcal{I}_c \quad I \preceq_1 C}{I :_1 C} \tag{8}$$

There are two possible ways of defining a type of an identifier. One way is to use the relationship $\preceq$. The other way is to use existent typing.

All possible types of I include the base types of I and all types that are more general than the base types. Note that the relationship $\preceq$ subsumes the relationship $\preceq_1$.

$$\frac{I \in \mathcal{I}_i \quad C \in \mathcal{I}_c \quad I \preceq C}{I : C} \tag{9}$$

The bridge between the typing relation and subtype relation is provided by adding a new typing rule [5]. The following rule is called *rule of subsumption*.

$$\frac{I \in \mathcal{I}_i \quad I : S \quad S \preceq T}{I : T} \tag{10}$$

3.3 Intersection type

The instances of the intersection type $T_1 \wedge T_2$ are objects belonging to both T_1 and T_2. The type $T_1 \wedge T_2$ is the greatest lower bound of the types T_1 and T_2. In general, $\wedge[T_1 \ldots T_n]$ is the greatest lower bound of types $T_1 \ldots T_n$ [3, 4].

$$T_1 \wedge T_2 \preceq T_1 \tag{11}$$

$$T_1 \wedge T_2 \preceq T_2 \tag{12}$$

$$\wedge[T_1 \ldots T_n] \preceq T_i \tag{13}$$

If the type S is more specific than the types $T_1 \ldots T_n$ then S is more specific then $\wedge[T_1 \ldots T_n]$. First, we present the rule for a pair of types T_1 and T_2.

$$\frac{S \preceq T_1 \quad S \preceq T_2}{S \preceq T_1 \wedge T_2} \tag{14}$$

$$\frac{\text{forall i, } S \preceq T_i}{S \preceq \wedge[T_1 \ldots T_n]} \tag{15}$$

3.4 Union type

The intersection and union types are dual. This can be seen also from the rules that are used for each particular type.

The instances from the union type $T_1 \vee T_2$ are either the instances of T_1 or T_2, or the instances of both types. The type $T_1 \vee T_2$ is the smallest upper bound of the types T_1 and T_2. In general, $\vee[T_1 \ldots T_n]$ is the smallest upper bound of types $T_1 \ldots T_n$ [2].

$$T_1 \preceq T_1 \vee T_2 \tag{16}$$

$$T_2 \preceq T_1 \vee T_2 \tag{17}$$

$$T_i \preceq \vee[T_1 \ldots T_n] \tag{18}$$

If the type T is more general than the types $S_1 \ldots S_n$ then T is more general then $\vee[S_1 \ldots S_n]$. First, we present the rule for types T_1 and T_2.

$$\frac{S_1 \preceq T \quad S_2 \preceq T}{S_1 \vee \S_2 \preceq T} \tag{19}$$

$$\frac{\text{forall i, } S_i \preceq T}{\vee[S_1 \dots S_n] \preceq T} \tag{20}$$

3.5 Type-checking triples

Triples and schema triples.

– *Is the following defined in formalization of KG?*
– *Maybe typing of ground, schema triples is presented? Which aspect?*
– *Show the complete poset of triples.*
– *Define the set of ground triples.*
– *Define the set of type triples (schema triples) and the schema graph .*
– *Define the stored schema graph.*

Deriving a base type of a triple. The base type of an individual identifier i is a class c related to i by one-step relationship $\preceq_1$. In terms of the concepts of a knowledge graph, c and i are related by the relationship rdf:type.

A base type of a triple $t = (s, p, o)$ is a triple $T = (T_s, T_p, T_o)$ that includes the base types of t's components. A base type of a triple is defined by the following rule.

$$\frac{s :_1 T_o \quad p :_1 T_p \quad T_p \preceq \text{rdf:Property} \quad o :_1 T_o}{(s, p, o) :_1 T_s * T_p * T_o} \tag{21}$$

The types of s and o can be any classes T_s and T_o from $\mathcal{I}_c$, while the type of p has to be a class T_p that is a subclass of rdf:Property. The typing of a triple t is correct since the interpretation of T includes t. Moreover, the types T that are derived by the above rule are minimal in the sense that given the information provided, i.e., the types of t's components, their interpretations are minimal possible comparing them to the interpretations of all other derived types of t.

Deriving a top type of a triple. The following rule is a judgment for a top type of a concrete triple $t = (s, p, o)$. A top type of a triple t is the most specific type from the stored schema graph which interpretation includes t.

We first find the schema triples for a given predicate p. The set of stored schema triples is constructed by selecting the most specific schema triples with a predicate that is more general then p.

$$\begin{aligned} S_0 = \{(T_s, p', T_o) \mid p' \succeq p \wedge (p', \text{dom}, T_s) \in g \wedge (p', \text{rng}, T_o) \in g \wedge \\ \nexists p''(p'' \preceq p' \wedge (p'', \text{dom}, T_s) \in g \wedge (p'', \text{rng}, T_o) \in g)\} \end{aligned} \tag{22}$$

Generator view of rules: Just describe the properties of pre-conditions and conclusions.

$$\frac{T \in \text{ssg} \quad p \preceq T_p \quad \text{for all } T' \in \text{ssg},\ T' \succ T \vee p \succ T'_p \vee T'_p \succ T_p}{(s,p,o) :_2 T} \tag{23}$$

$$\frac{T \in \text{ssg} \quad t :_1 T_1 \quad T_1 \preceq T \quad \not\exists S \in \text{ssg},\ S \prec T \wedge T_1 \preceq S}{(s,p,o) :_2 T} \tag{24}$$

The first two premises require that the type T is an element of the stored schema graph, and the predicate of T, i.e., T_p, is more general than the predicate p of the input triple (s, p, o).

The last premise in the above rule requires that the top type T is the least general type including a predicate equal or more general to p. The condition can be better understood in the existential form: $\not\exists T' \in \text{ssg} : T' \preceq T \wedge p \preceq T'_p \preceq T_p$.

Note that the rule is not linked to the t's components s and o in any way. This means that $s \preceq T_S$ and $o \preceq T_O$ may not hold.[3]

From the other point of view, the schema triples are obtained from the inherited values of the predicates rdfs:domain and rdfs:range. The inherited values have to be the closest when traveling from property p towards the more general properties. Note that multiple different schema triples are possible only in the case of multiple inheritance.

Typing a triple.

– Why the type selected from ssg?
– How (conceptually) types from ssg are selected?

The type of a triple $t = (s, p, o)$ is computed by first deriving the base type T and the top type S of t. Then, we check if S is reachable from T through the sub-class and sub-property hierarchies, i.e., $T \preceq S$.

$$\frac{(s,p,o) :_1 T \quad (s,p,o) :_2 S \quad (T \preceq S)}{(s,p,o) : S} \tag{25}$$

– How to compute $T \preceq S$? Refer to position where we have a description.
– How to gather a complete type of t including different $S \in sgg$? Union of selected S's... this is a complete *type. It does make sense.*

– Order the possible derivations, gatherings (groupings) ... of types.
– How to derive all possible types of a triple? How to integrate them using union and intersection types?
– How to derive types of a triple deriving in some specific direction? For example, the cover (lub) type of a triple? The most specific type (base type)?

– Possible diagnoses.
– Components not related to a top type of a triple?
– Components related to sub-types of a top type?

[3] Does it make sense to add the conditions? Further, at this point the type errors can be catched.

– *Above pertain to all components.*

3.6 Typing a graph.

– *What is a type of a graph?*
– *A type of a graph is a graph!*
– *It includes a set of schema triples forming a schema graph.*

– *Typing a graph bottom-up?*
– *Checking that all the triples are of correct types.*

Typing a schema triple.

– *What can be checked?*
– *Is a schema triple properly related to the super-classes and types of components.*
– *Consistency of the placement of a class in an ontology. What is this?*
– *A class or predicate component not related to other classes?*
– *A class or predicate component attached to "conflicting" set of classes? What can be detected?*
– *@kiyoshi Do you see any other examples?*
–

4 Empirical analysis

5 Conclusions

References

1. A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4), jul 2021.
2. B. C. Pierce. Preliminary investigation of a calculus with intersection and union types, 1990.
3. B. C. Pierce. Programming with intersection types, union types, and polymorphism, 1991.
4. B. C. Pierce. Intersection types and bounded polymorphism, 1996.
5. B. C. Pierce. *Types and Programming Languages*. MIT Press, 1 edition, Feb. 2002.