Type-checking knowledge graphs

Iztok Savnik[1]

Faculty of mathematics, natural sciences and information technologies,
University of Primorska, Slovenia
iztok.savnik@upr.si

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 Definition of knowledge graph

This section defines a knowledge graph as a RDF graph [6] using RDF-Schema [7] for the representation of the structural part of a knowledge base.

Let I be the set of URI-s, B be the set of blanks and L be the set of literals. Let us also define sets $S = I \cup B$, $P = I$, and $O = I \cup B \cup L$.

Let I be the set of URI-s, B the set of blanks and L be the set of literals. Let us also define sets $S = I \cup B$, $P = I$, and $O = I \cup B \cup L$.

RDF triple is a triple $(s, p, o) \in S \times P \times O$. *RDF graph* $g \subseteq S \times P \times O$ is a set of triples. Set of all graphs will be denoted as G. We suppose the existence of a set of variables V and the set of *terms* $T = O \cup V$. Term $t \in T$ is ground if $t \in O$.

We say that RDF graph g_1 is *sub-graph* of g_2, denoted $g_1 \sqsubseteq g_2$, if all triples in g_1 are also triples from g_2.

– Define major structure of KG on the basis of the sorts of data.

– ...the set I includes individual identifiers I_i, class identifiers I_c and predicate identifiers I_p.

3 Type system used

3.1 Product types

3.2 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, $\wedge[T_1 \dots T_n]$ is the greatest lower bound of types $T_1 \dots T_n$ [3, 4].

$$T_1 \wedge T_2 \preceq T_1 \tag{1}$$

$$T_1 \wedge T_2 \preceq T_2 \tag{2}$$

$$\wedge[T_1 \dots T_n] \preceq T_i \tag{3}$$

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

$$\frac{S \preceq T_1 \quad S \preceq T_2}{S \preceq T_1 \wedge T_2} \tag{4}$$

$$\frac{\text{forall i, } S \preceq T_i}{S \preceq \wedge[T_1 \dots T_n]} \tag{5}$$

3.3 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, $\vee[T_1 \dots T_n]$ is the smallest upper bound of types $T_1 \dots T_n$ [2].

$$T_1 \preceq T_1 \vee T_2 \tag{6}$$

$$T_2 \preceq T_1 \vee T_2 \tag{7}$$

$$T_i \preceq \vee[T_1 \dots T_n] \tag{8}$$

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

$$\frac{S_1 \preceq T \quad S_2 \preceq T}{S_1 \vee \S_2 \preceq T} \tag{9}$$

$$\frac{\text{forall i, } S_i \preceq T}{\vee[S_1 \dots S_n] \preceq T} \tag{10}$$

4 Typing identifiers

– Introduction includes the formalization of RDF, RDF-Schema as given in Angles and Peres.
– Typing idents without considering and info about the triples.

– General.
– At the end of section define the lub type using $\wedge\vee$ types.
– 1. Define lub types as the closest to base types of given ground ident.
– 2. Collect all lub types using $\wedge$ type in a single type.

– Details.
– 1. First define base type of identifiers $:_1$ and stored subtyping relationship $\preceq_1$.
– 2. From the basis define the indent typing $:$ and subtyping rel $\preceq$ among identifiers.
– 3. Include the link between subtyping and typing.
– 4. Define lub type using $\wedge$ type for a given ground ident.

4.1 Typing literals

– Literals are identifiers of atomic type!

4.2 Stored typing and subtyping of identifiers.

– Partial ordering defined with stored schema triples in a database.
– The relationships that poset $\preceq_1$ covers are rdfs:subClassOf and rdfs:subPropertyOf.
– Identifiers included in $:_1$ are between ground idents and base classes.
– This allows us to separate and also address separately the ssg and subtyping relationship.

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

Reflecting the one-step relationship rdf:type with $:_1$.

$$\frac{I_1 \in \mathcal{I}_i \quad I_2 \in \mathcal{I}_c \quad (I_1, \text{rdf:type}, I_2) \in \mathcal{D}}{I_1 :_1 I_2} \tag{11}$$

Expressing the one-step subtyping relationship rdfs:subClassOf with $\preceq_1$.

$$\frac{I_1, I_2 \in \mathcal{I}_c \quad (I_1, \text{rdfs:subClassOf}, I_2) \in \mathcal{D}}{I_1 \preceq_1 I_2} \tag{12}$$

Expressing the one-step subtyping relationship rdfs:subPropertyOf with $\preceq$.

$$\frac{I_1, I_2 \in \mathcal{I}_p \quad (I_1, \text{rdfs:subPropertyOf}, I_2) \in \mathcal{D}}{I_1 \preceq_1 I_2} \tag{13}$$

4.3 Typing and subtyping identifiers.

– Extend one-step relationships $:_1$ and $\preceq_1$ to relationships $:$ and $\preceq$.
– The relation $\preceq$ is the relation $\preceq_1$ extended with the reflexivity and transitivity.
– Link typing $:$ and subtyping $\preceq$ with a rule.

Generalizing one-step relationship $\preceq_1$ to the relationship $\preceq$.

$$\frac{I_1, I_2 \in \mathcal{I} \quad I_1 \preceq_1 I_2}{I_1 \preceq I_2} \tag{14}$$

Subtyping is reflexive.

$$\frac{S \in \mathcal{I}}{S \preceq S} \tag{15}$$

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.

$$\frac{S, U, T \in \mathcal{I} \quad S \preceq U \quad U \preceq T}{S \preceq T} \tag{16}$$

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

$$S \preceq \top \tag{17}$$

A base type of an individual identifier I is a type of I.

$$\frac{I \in \mathcal{I}_i \quad C \in \mathcal{I}_c \quad I :_1 C}{I : C} \tag{18}$$

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

$$\frac{I \in \mathcal{I}_i \quad I : S \quad S \preceq T}{I : T} \tag{19}$$

– Properties have dual role: they are instances and types at the same time.
– Present the features of properties from this point of view.

– Put together the base types of ground identifiers using $\wedge$ type.
– First, the base type of an ground identifier is the $\wedge$ of all base types.
– The base type *of a ground identifier is defined explicitely!*

– The lub types of base types B are the smallest types related to all base types.
– The lub types are related by $\wedge$ to form a (complete) type of a ground identifier.
– The complete type of a ground identifier is defined explicitely.

5 Typing triples

– There are two basic aspects of a triple type.
– First, the type is computed bottom-up: from the stored types of triple components.
– Second, the type can be computed top-down: from the user-defined domain/range types of properties.

– Ground type of a triple is computed first using $:_1$.
– Next, the lub type of a triple is derived using $:_l$.
– From the top side of the ontology, the stored type $:_s$ is determined based on p.
– Finally, the type $:$ of t is deterimned by summing alternative $:_s$ types.

– Interactions between the $\wedge/\vee$ types of triple components and triples must be added.
– Analogy between. the types of functions in lambda calculus and types of triples.
– Show rules relating $\wedge/\vee$ types and triple types. Example.
– E.g., $(S_1 \wedge S_2) * p * R = S_1 * p * R \wedge S_2 * p * R$.

5.1 Deriving a base type of a triple.

A base type of a triple $t = (s, p, o)$ is a triple $T = T_s * p * T_o$ that includes the base types of t's components s and o, and the property p which now has the role of a type. A base type of a triple is defined by the following rule.

$$\frac{t = (s, p, o) \quad s :_1 T_s \quad p :_1 \text{owl:ObjectProperty} \quad o :_1 T_o}{t :_1 T_s * p * T_o} \tag{20}$$

The type of S component T_s is one of the ground types of s, and the type of O components is one of the ground types of o. The predicate P is treated differently to S and O components. The predicates have the role of classes while they are instances of owl:ObjectProperty.

There are multiple ground types of a triple. They can be gathered into a single type using the $\wedge$ type.

$$\frac{t \in \mathcal{T}_i \quad \forall\, i, t :_1 T_i}{t :_\wedge \bigwedge[T_1..T_n]} \tag{21}$$

– Ground typing by using lub types of $T_1..T_n$.
– Is this really needed?

Let us now select the least upper bound types of the ground types derived by the rule 20.

$$\frac{t :_1 \bigwedge[T_1..T_n] \quad \forall i, T_i \preceq T}{\vdash t :_{\wedge 1} T} \tag{22}$$

– Now only the minimal types have to be selected.

$$\frac{t :_{\wedge 1} T \quad \forall (t :_\wedge S), S \preceq T}{t :_{\wedge 2} T} \tag{23}$$

– Finally, lub2 types are gathered with $\wedge$.

$$\frac{\forall i, t :_{\wedge 2} T_i}{t :_\wedge T} \tag{24}$$

5.2 Stored types of triples.

– Make a bit more clear picture of the GLB triple types selected as the final stored type of a triple.
– Analysis tool. Show minimality of the stored types (either enumerated of gathered with $\bigvee$).
– Reminder: *when a complete stored (user-defined) type is related to the base type of a triple, some of GLB types may be eliminated.*

– Predicates should be treated in the same way as the classes.
– They can have a rich hierarchy.
– Note: *Where to include discussion on special role of predicates and their relations to classes?*

We first find a stored schema triples for a given triple $t = (s, p, o)$. A stored schema triple is constructed by selecting types including a predicate that have the domain and range defined.

$$\frac{t \in \mathcal{T}_i, t = (s, p, o) \quad p \preceq p' \quad (p', \text{domain}, T_s) \in g \quad (p', \text{range}, T_o) \in g}{t :_s (T_s, p, T_o)} \tag{25}$$

– Comments and description of the above rule.

The domain and range of a predicate p can be defined for any super-predicate, they do not need to be defined for p. In addition, the domain and range of a predicates do not need to be defined for the same predicate; they can be defined for any of the super-predicates separately. The following rule captures also the last statement.

$$\frac{t = (s, p, o) \quad p \preceq p_1 \quad p \preceq p_2 \quad (p_1, \text{domain}, T_s) \in g \quad (p_2, \text{range}, T_o) \in g}{t :_s (T_s, p, T_o)} \tag{26}$$

– If p inherits from multiple $p' \succeq p$, then the above rule generates multiple types. Explain.

Note that the type is determined only in the case that the domain and range of p or some $p' \succeq p$ is defined. Otherwise, the domain and range shoud be $\top$.

The following rule is a judgment for a (user-defined) type of a concrete triple $t = (s, p, o)$. A user-defined type of t is the greatest lower bound of stored types generated by the rule 26.

– A general explanation for choosing glb type should be given. Further expanded below. (?)

$$\frac{t \in \mathcal{T}_i \quad t :_s T \quad \forall S, t :_s S \wedge T \preceq S}{t :_\vee T} \tag{27}$$

The first premise says that t is an individual triple. The second premise determines one of the stored type T of t. The third premise requires that the user-defined type T is the least general type.

The implementation view of the above rule is as follows. 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.

– Why do we have multiple glb types? Add explanation.
– (The following paragraph is not completed! More precise!)

Multiple different schema triples for a given t are possible only in the case of multiple inheritance, in the case of the definition of the disjunctive domain/range types, or

if predicate is defined for semantically different concepts. (describe each possibility in more detail.)

– Add explanation about why glb types are connected with $\vee$.
– $\vee$ is used since they represent alternative semantics of a triple.
– When type-checking a language, a context can be used to resolve disjunctions.

Now we can put together the greatest lower bounds of the top types by making the union of the glb types computed by the previous rule.

$$\frac{t \in \mathcal{T}_i \quad \forall i, t :_\vee T_i}{t :_\vee \bigvee T_i} \tag{28}$$

– Expand on the influence/effect of using $\bigvee$ type in context and in other rules.

5.3 Typing a triple.

– Two ways of defining semantics.
– 1) enumeration style: stored types are enumerated as alternatives ($\vee$).
– 2) packed together: alternative types are packed in one $\bigvee$ type.
– One advantage of (1) is that individual glb types can be processed further individually.
– Advantage of (2) is the higer-level semantics without going in implementation.

– Now stored types have to be related to all lub base types to represent the correct type of a triple.
– It seems it would be easier to check the pairs one-by-one using (1) in algorithms.
– In case of using complete types in the phases, types would further have to be processed by $\wedge, \vee$ rules.

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., $T \preceq S$.

$$\frac{(s, p, o) :_1 T \quad (s, p, o) :_2 S \quad (T \preceq S)}{(s, p, o) : S} \tag{29}$$

– How to compute $T \preceq S$? Refer to position where we have a description.
– Order the possible derivations, gatherings (groupings) ... of types.

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

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.

6.1 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?
–

7 Empirical analysis

8 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.
6. Resource description framework (rdf). http://www.w3.org/RDF/, 2004.
7. Rdf schema. http://www.w3.org/TR/rdf-schema/, 2004.