neXSim: A System for Explaining Nexus of Similarities*

Giovanni Amendola[1,*,†], Pietro Cofone[1,*,†], Marco Manna[1,*,†] and Aldo Ricioppo[1,2,*,†]

[1] *Department of Mathematics and Computer Science, University of Calabria, Italy*

[2] *Department of Computer Science, University of Cyprus, Cyprus*

Abstract

Recognizing and explaining nexus of similarities between entities is a crucial task that arises in numerous real-life scenarios. Understanding why a group of individuals shares a certain disease or why some products may be more appealing than others are natural examples. Over the last thirty years, a variety of logic-based approaches have been proposed to address this task. However, despite these efforts, there is a gap in available systems capable of exploiting real-world knowledge bases. In response to this gap, the paper presents neXSim, a prototypical web platform based on a recent logic-based framework explicitly designed to be suitable for a practical and effective implementation. This platform, instantiated over BabelNet 4.0, is capable of exploiting very large knowledge bases on-the-fly and in real-time, while providing formal, concise, and human-readable explanations.

Keywords

logic-based reasoning, knowledge base integration, semantic similarity, real-time reasoning systems, algorithms implementation

1. Introduction

The task of recognizing and expressing commonalities between entities within a knowledge base has captured the interest of researchers across various disciplines over the past three decades. This exploration spans diverse AI communities, ranging from Description Logics to Semantic Web and Database Theory [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. Existing approaches vary across four key dimensions: the form of the input (e.g., pairs of entities, sets of entities, sets of entity tuples), the type of knowledge bases they can handle (e.g., DL knowledge bases, RDF, relational databases), the scope of knowledge considered for each input item (e.g., the entire knowledge base or selected excerpts), and the specific formalism to express commonalities (e.g., DL Concepts, r-graphs, (U)CQs, SPARQL queries, rooted-CQs).

Despite the significant research efforts, there remains a gap in available online systems capable of addressing the considered reasoning task in real-world scenarios exploiting very large knowledge graphs. In response to this gap, the paper presents neXSim, a prototypical web platform based on a recent logic-based framework [11] explicitly designed to be suitable for a practical and effective implementation. In particular, this framework: (*i*) can accommodate different types of underlying knowledge bases; (*ii*) adopts a notion of *summary selector* to focus on relevant knowledge about the input items; (*iii*) is conceived to handle sets of entity tuples; and (*iv*) expresses *nexus of similarities* (i.e., commonalities) via *nearly connected conjunctive formulas* (essentially, rooted-CQs plus constants). Overall, it guarantees that, for any input items, there always exists a finite formula expressing all commonalities, called *characterization.* Moreover, to guarantee concise and human-readable explanations, in the same spirit as done also for rooted-CQs [10], the framework promotes the adoption of size-wise minimal formulas among all equivalent characterizations.

To illustrate our approach, consider a Knowledge Base (KB) describing *Leonardo da Vinci* as a painter, sculptor, artist, and person of Italian nationality who designed war machines, while it describes

RCRA 2025: 32nd Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion.

[*] Corresponding author.

[†] These authors contributed equally.

giovanni.amendola@unical.it (G. Amendola); pietro.cofone@unical.it (P. Cofone); marco.manna@unical.it (M. Manna); aldo.ricioppo@unical.it (A. Ricioppo)

0000-0002-2111-9671 (G. Amendola); 0009-0003-6262-4732 (P. Cofone); 0000-0003-3323-9328 (M. Manna); 0009-0007-5911-6015 (A. Ricioppo)

Michelangelo as a painter, sculptor, artist, and person of Italian nationality who wrote sonnets. Our first goal is to capture the nexus of similarity shared by them via a so called *core characterization* [11]. The latter, in essence, is the most compact, non-redundant formula that is still exhaustive in listing all shared properties. From what we know about the two entities, the core characterization is:

$$\mathsf{x} \leftarrow \mathsf{isA(x, Painter), isA(x, Sculptor), isA(x, Artist), isA(x, Person), nationality(x, Italian).}$$

It is crucial to note that, at this stage, no information is omitted based on logical entailment. The objective here is to establish the complete ground truth, expressed in its most concise syntactical form. With this notion of explanation as our target, this paper introduces neXSim, a research prototype instantiated over BabelNet 4.0 [12] and available at https://tinyurl.com/bdhpvdc7. In the context of Explainable AI, neXSim is designed to compute such core characterizations for a broad audience. This capability serves not only human users—from domain experts formalizing knowledge to curious users exploring a topic—but also automated agents like recommender systems that must justify their suggestions. Accordingly, the tool allows users to assemble a group of entities with ease, either by searching for them via keywords or by providing their specific identifiers. The main contributions of this paper are therefore twofold. First, in Section 3, we provide empirical evidence that core characterizations are a superior alternative to *canonical characterizations* [11]. These canonical characterizations are built using classical techniques drawn from various fields such as database theory, Query-by-Example, and logic definability. While such methods are computationally appealing, they often lead to verbose and redundant results that are ill-suited for end-users. We show that, in practical scenarios, our approach produces characterizations that are significantly more compact while remaining just as complete. Second, in Section 4 we introduce *kernel explanations* (*ker*) as a subsequent refinement step, designed for maximum readability by pruning logically deducible facts from the core characterization. Revisiting our example, the complete core characterization is refined into a *ker* explanation:

$$\mathsf{x} \leftarrow \mathsf{isA(x, Painter), isA(x, Sculptor), nationality(x, Italian).}$$

This is achieved by observing that, in accordance with the information in the KB that we have defined, being a painter implies being an artist, which in turn implies being a person, making the latter two concepts redundant for a human reader. We then detail how neXSim efficiently computes these explanations, and envision future work to further enhance their user-friendliness, for instance by generating them in natural language.

2. Existing Framework

We outline the recent framework for characterizing nexus of similarities [11, 13].

2.0.1. Basics.

An *atom* is a labelled tuple $p(t_1, ..., t_n)$, where p is a *predicate* and each t_i is a *term*, either a *constant* or a *variable*. A *structure* is a set of atoms. A *dataset* is a variable-free structure. Constants are called *entities* in datasets encoding knowledge graphs. A *homomorphism* from a structure S to a structure S', denoted $S \longrightarrow S'$, is a map h from $terms(S)$ to $terms(S')$, where $\alpha = p(t_1, ..., t_n) \in S$ implies $h(\alpha) = p(h(t_1), ..., h(t_n)) \in S'$ and $h(c) = c$ for each constant c. A *(conjunctive) formula* $\varphi(x_1, ..., x_n)$ is an expression $x_1, ..., x_n \leftarrow p_1(\mathbf{t}_1), ..., p_m(\mathbf{t}_m)$, where n is its *arity*, m is its *size*, each $\mathbf{t}_i$ is a sequence of terms, each $p_i(\mathbf{t}_i)$ is an atom, and each x_j is a (*free*) variable occurring in some of the atoms of φ. The *output* of φ over a dataset D is the set $\varphi(D)$ of every tuple $\langle t_1, ..., t_n \rangle$ such that $atoms(\varphi) \longrightarrow D$ via a homomorphism ensuring that each $h(x_i) = t_i$.

2.0.2. Selective knowledge bases.

A *knowledge base* (KB) is a pair $K = (D, O)$, where D is a dataset and O is an (onto)logical first-order theory [14, 15]. As is customary, an atom α is *entailed* by K if it occurs in every model of K. The set of

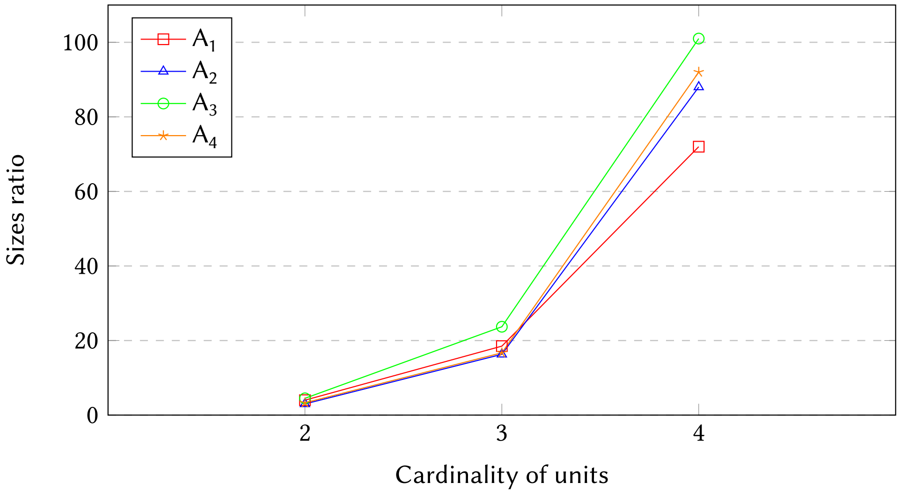

Figure 1: Sizes ratio between canonical and core characterizations.

atoms entailed by K is denoted as $ent(K)$. A *selective knowledge base* (SKB) is a pair $\mathcal{S} = (K, \varsigma)$, where K is a KB and ς is a *summary selector*, namely and algorithm that takes in input K together with a tuple τ of constants and returns a dataset $\varsigma(K, \tau) \subseteq ent(K)$ representing a *summary* of τ (w.r.t. K). When K is clear from the context, we write $\varsigma(\tau)$ instead of $\varsigma(K, \tau)$. Angle brackets may be omitted for unary tuples.

2.0.3. Explanation language.

A *nearly connected formula* (NCF) is a formula $\varphi(x_1, ..., x_n)$ such that the structure $atoms(\varphi) \cup \{\text{dummy}(x_1, ..., x_n)\}$ is connected.[1] Intuitively, each atom of φ can "reach" some free variable. For example, $\text{x} \leftarrow \text{isA}(\text{x}, \text{ap}), \text{located}(\text{x}, \text{y}), \text{partOf}(\text{y}, \text{US})$ and $\text{x}, \text{y} \leftarrow \text{isA}(\text{x}, \text{ap}), \text{partOf}(\text{y}, \text{US})$ are both nearly connected, but $\text{x} \leftarrow \text{isA}(\text{x}, \text{ap}), \text{partOf}(\text{y}, \text{US})$ is not. An *instance* of φ according to an SKB $\mathcal{S} = (K, \varsigma)$ is any tuple τ of constants that belongs to $\varphi(\varsigma(\tau))$. The set of all instances of φ is denoted by $inst(\varphi, \mathcal{S})$. Clearly, $inst(\varphi, \mathcal{S}) \subseteq \varphi(ent(K))$. Given two NCFs $\varphi_1(x_1, ..., x_n)$ and $\varphi_2(y_1, ..., y_n)$, by $\varphi_1 \longrightarrow \varphi_2$ we mean that $atoms(\varphi_1) \longrightarrow atoms(\varphi_2)$ via a homomorphism h ensuring that each $y_i = h(x_i)$.

2.0.4. Characterizations.

Fix an SKB $\mathcal{S} = (K, \varsigma)$ and a set $\mathbf{U} = \{\tau_1, ..., \tau_m\}$ of n-ary tuples of constants, called (n-ary) *unit*. An NCF φ *explains* (some nexus of similarity between the tuples of) $\mathbf{U}$ if $inst(\varphi, \mathcal{S}) \supseteq \mathbf{U}$, whereas φ *characterizes* (the nexus of similarity between the tuples of) $\mathbf{U}$ if φ explains $\mathbf{U}$ and $\varphi' \longrightarrow \varphi$ whenever φ' explains $\mathbf{U}$. Intuitively, φ characterizes $\mathbf{U}$ if it encompasses the nexus of similarities expressed by any other NCF explaining $\mathbf{U}$. To ensure the existence of characterizations, we assume that both $ent(K)$ and any $\varsigma(\tau)$ contain a special atom $\top(c)$ for each of their constants c. By adapting the well-known direct product operator between structures and taking into account the properties of NCFs (presence of constants and nearly connectedness), it is possible to build a *canonical characterization* of $\mathbf{U}$, denoted as $can(\mathbf{U}, \mathcal{S})$ as described in [11]. As it will be clearer in Section 3, $can(\mathbf{U}, \mathcal{S})$ may contain redundant atoms. They, however, can be eliminated by keeping only a *core*[2] of the atoms of $can(\mathbf{U}, \mathcal{S})$ preserving all free variables. This (generally) smaller characterization is denoted by $core(\mathbf{U}, \mathcal{S})$.

3. Canonical vs Core Characterizations

The objective of this section is to demonstrate that, even within small knowledge bases, core characterizations are preferable in terms of explainability, readability, and conciseness compared to canonical

[1] Inductively, a set of atoms is *connected* if it is a singleton or can be partitioned into two connected sets sharing at least one term.

[2] A *core* of a structure S is any $S' \subseteq S$ such that $S \longrightarrow S'$ and there is no $S'' \subset S'$ such that $S' \longrightarrow S''$.

characterizations.

From a theoretical viewpoint, we know there are families $\{(\mathbf{U}_i, \mathcal{S}_i)\}_{i>1}$ of SKBs and units of size proportional to i, for which the sizes of $can(\mathbf{U}_i, \mathcal{S}_i)$ and $core(\mathbf{U}_i, \mathcal{S}_i)$ coincide [11]. But, there are families where the size of $can(\mathbf{U}_i, \mathcal{S}_i)$ is exponential in i, while the size of $core(\mathbf{U}_i, \mathcal{S}_i)$ is constant. As an example, consider $\mathbf{U}_i = \{\mathrm{a}_1, ..., \mathrm{a}_i\}$, $\mathcal{S}_i = (K_i, \varsigma_i)$, $K_i = (D_i, \emptyset)$, $D_0 = \emptyset$, $D_i = \{\mathrm{p}(\mathrm{a}_i, \mathrm{b}_i),\ \mathrm{p}(\mathrm{a}_i, \mathrm{c}_i)\} \cup \bigcup_{0<j<i} D_j$, and $\varsigma_i(d, K_i) = \{\mathrm{p}(d, e), \top(e) \mid \mathrm{p}(d, e) \in D_i\} \cup \{\top(d)\}$. In particular, $|can(\mathbf{U}_i, \mathcal{S}_i)| = 1 + 2^{i+1}$, while $|core(\mathbf{U}_i, \mathcal{S}_i)| = 3$. It is therefore important to understand, in practical cases, the actual ratio between the two forms of characterizations. To this aim, in what follows, we present an excerpt from an experimental analysis conducted on a small real-world knowledge base in the movies domain.

The considered knowledge base comprises data on 130 notable films from the past 30 years. It also includes profiles of influential industry figures. The dataset's structure is based on binary relations, connecting two entities through predicates such as directedBy or starring, to represent facts within the domain. In particular, it comprises 399 entities, 1,273 binary relations as extensional knowledge, and an ontology consisting of 13 rules: one for the transitive closure of the isA relation, and the others to derive roles of the entities (e.g., director or writer). On average, each entity has 15 items in its summary. Then, we randomly generated and tested 10,000 units for each cardinality between 2 and 4.

For each size of the considered units, we analyzed the trends about the ratio between the sum of the sizes of all canonical characterizations and the sum of the sizes of all core characterizations under four different assumptions: (A_1) all units; (A_2) only units where the instances of their characterizations do not coincide with all possible entities; (A_3) as in (A_1) but removing $\top$-atoms from characterizations; (A_4) as in (A_2) but removing $\top$-atoms from characterizations. As it will be clear in Section 4.4, the last two trends have been considered since $\top$-atoms can generally and safely be omitted for readability. Figure 1 illustrates the essence of our findings: overall, the size of canonical characterizations becomes more and more larger than the one of core characterizations when considering units of increasing size; indeed, all four considered trends are more than linear. We close by underlining that, while the maximum size of canonical characterizations was 9,705 (resp., 4,852) with (resp., without) $\top(X)$ atoms, the maximum size of core characterizations was of 29 (resp., 15). Full details are available at https://tinyurl.com/3sks2tax.

4. From theory to practice with neXSim

This section introduces neXSim 0.1-alpha, the initial prototype system implementing the framework outlined in the previous section. Currently, the system is capable of dealing with unary units of arbitrary size, characterized according to SKBs that are defined from RDF knowledge graphs (KGs) paired with simple ontological Datalog rules. As a proof of concept, neXSim has been instantiated with BabelNet 4.0.1, a well-known generalist RDF KG [12].

4.1. From RDF Knowledge Graphs to SKBs

From a given RDF KG G, we illustrate the shape of the SKB $\mathcal{S} = (K, \varsigma)$ with $K = (D, O)$ that neXSim conceptually associates to G at present. For each triple (s, R, o) in G, the dataset D contains the atom $R(s, o)$. Additionally, if R is of a *hypernymy* kind (e.g., isA) or of a *holonymy* kind (e.g., partOf), then O contains the following rule: $R(\mathrm{x}, \mathrm{z}) \leftarrow R(\mathrm{x}, \mathrm{y}), R(\mathrm{y}, \mathrm{z})$. Clearly, for example, if both $\mathsf{isA}(a, b)$ and $\mathsf{isA}(b, c)$ belong to D, then both of them, together with $\mathsf{isA}(a, c)$, belong to $ent(K)$. In what follows, we assume that relations of a hypernymy or a holonymy kind are acyclic. When this is not the case, G has to be cleaned or repaired in advance. Finally, for each entity e occurring in D, we have that $\varsigma(e)$ consists of the set of atoms $\{p(e, c), \top(e), \top(c) \mid p(e, c) \in ent(K)\}$.

The framework illustrated in Section 2 may accommodate alternative representations. Indeed, one could encode a triple (s, R, o) in G as the atom $\mathsf{triple}(s, R, o)$. Consider, however, the triples (a, isA, person), (a, founded, Google), (b, isA, person), (b, uses, Google), the unit $\mathbf{U} = \{\mathrm{a}, \mathrm{b}\}$, and σ as above. If we encode triples as binary atoms, then a characterization of $\mathbf{U}$ looks like $\mathrm{x} \leftarrow \mathsf{isA}(\mathrm{x}, \mathrm{person})$; whereas, if we encode triples as ternary atoms, then it looks like $\mathrm{x} \leftarrow \mathsf{triple}(\mathrm{x}, \mathsf{isA}, \mathrm{person}), \mathsf{triple}(\mathrm{x}, \mathrm{y}, \mathrm{Google})$. Since we believe that the latter in not extremely more informative, we currently prefer simpler explanations.

4.2. Dealing with Hypernyms

Fix an SKB $\mathcal{S} = (K, \varsigma)$ as illustrated in Section 4.1, including the binary predicate isA and the ontological rule $\mathsf{isA(x, z)} \leftarrow \mathsf{isA(x, y), isA(y, z)}$. Consider a scenario where we have a unit $\mathbf{U} = \{e_1, e_2\}$, with both entities being students. Since $\mathsf{isA(student, person)}$ belongs to $ent(K)$, it is reasonable to avoid considering that both entities of $\mathbf{U}$ are also persons. Hence, from $core(\mathbf{U}, \mathcal{S})$, we could chose to "hide" the atom $\mathsf{isA}(x, \mathsf{person})$, as it may be obvious with respect to $\mathcal{S}$, while displaying $\mathsf{isA}(x, \mathsf{student})$. By following this intuition, we can both speed up the computation of nexus explanations and improve the user experience. To achieve this, we define a suitable variant of ς, called $\tilde{\varsigma}$.

An *ancestor* of an entity e is any a such that $\mathsf{isA}(e, a)$ belongs to $ent(K)$. Consider a unary unit $\mathbf{U} = \{e_1, ..., e_m\}$. A *common ancestor* for $\mathbf{U}$ is any entity a that is an ancestor of each entity in $\mathbf{U}$. A *least common ancestor* for $\mathbf{U}$ is any common ancestor a such that, for each $\mathsf{isA}(c, a) \in ent(K)$, c is not a common ancestor for $\mathbf{U}$. We denote by $lca(\mathbf{U})$ the set of all least common ancestors for $\mathbf{U}$. For each $e \in \mathbf{U}$, we define $\varsigma_{\mathbf{U}}(e)$ as the set $\{p(e, c) \in \varsigma(e) \mid p \neq \mathsf{isA}\} \cup \{\mathsf{isA}(e, c) \in \varsigma(e) \mid c \in lca(\mathbf{U})\} \cup \{\mathsf{isA}(e, c) \in \varsigma(e) \mid p(e, c) \in \varsigma(e) \ \wedge \ p \neq \mathsf{isA}\}$. Intuitively, we retain the following: all atoms for non-hypernymy relations, the atoms containing the least common ancestors for $\mathbf{U}$, and the atoms containing common ancestors that also occur in non-hypernymy relations.

4.3. Virtualizing SKBs via Graph Databases

To efficiently deal with very large RDF-based SKBs as BabelNet, neXSim adopts the NoSQL graph database management system Neo4j [16]. In the rest of the section, fix an RDF graph G. Let $\mathcal{S} = (K, \varsigma)$ with $K = (D, O)$ be the SKB conceptually associated to G, as illustrated in Section 4.1. To create and on-the-fly navigate an instance I_G of Neo4j that allows to virtually interact with $\mathcal{S}$, we exploit the built-in query language of Neo4j called Cypher [17]. For example, to insert to I_G an atom $R(s, o)$ of D, we use the query:

```
MERGE (n1 {id:s})
MERGE (n2 {id:o})
MERGE (n1)-[:R]->(n2);
```

Moreover, to retrieve all entities directly connected to some entity e of D via some relation R, namely those in the set $\{a \mid R(e, a) \in D\}$, we can use the following path query:

```
MATCH ({id:e})-[:R]->(x)
WITH DISTINCT x.id as y RETURN y;
```

In particular, if R is a transitive relation (like isA or partOf), we can adapt the above path query, by replacing `[:R]` with `[:R*]`, resulting in a variable-length path query. If so, the result would now be the set $\{a \mid R(e, a) \in ent(K)\}$.

To compute $lca(\mathbf{U})$, we first add temporarily to both D and I_G a dummy atom $\mathsf{isA}(\heartsuit, e)$ for each $e \in \mathbf{U}$. Then, we exploit a Cypher procedure to retrieve the set $hyper(\mathbf{U}) = \{(\heartsuit, e) \mid \mathsf{isA}(\heartsuit, e) \in D\} \cup \{(s, o) \mid \mathsf{isA}(s, o) \in D \wedge \mathsf{isA}(\heartsuit, s) \in ent(K))\}$ as follows:

```
MATCH (node {id:♡})
CALL apoc.path.subgraphAll(node,
     {relationshipFilter:'isA>'})
YIELD nodes, relationships
UNWIND relationships as relation
WITH startNode(relation).id as x,
     endNode(relation).id as y,
     type(relation) as rel
WHERE rel = 'isA'
RETURN x, y;
```

Finally, from $hyper(\mathbf{U})$ it is possible to compute $lca(\mathbf{U})$ either directly in Python or via a simple set of Datalog rules with stratified negation [18] available https://tinyurl.com/3sks2tax.

Algorithm 1 Kernel of $\mathbf{U}$ according to $\mathcal{S} = (K, \varsigma)$

```
Input: $\varsigma_{\mathbf{U}}(e_1), ..., \varsigma_{\mathbf{U}}(e_m)$
Output: $ker(\mathbf{U}, \mathcal{S})$
$A := \varsigma_{\mathbf{U}}(e_1)$ and $a := e_1$
for $\ell \in \{2, ..., m\}$ do
    $(B, b) := (\varsigma_{\mathbf{U}}(e_\ell), e_\ell)$
    $(A, a) :=$ Prod&Core$(A, a, B, b)$
end for
return $a \leftarrow A$
```

Algorithm 2 Prod&Core

```
Input: $A, a, B, b$
Output: Core$(A \Diamond B)$, $a \Diamond b$
$v_1 := \{a \mapsto \mathrm{x}_{a,b}\}$ and $v_2 := \{b \mapsto \mathrm{x}_{a,b}\}$
$C :=$ predicates$(A) \cap$ predicates$(B)$
$A := v_1(A|_C)$ and $B := v_2(B|_C)$
Keep $:= \{p(s,o) \in A \cap B \mid \#_o(A \cup B) = 1\}$
$A := A \setminus \{p(s,o) \in A \mid \#_o(A) = 1\}$
$B := B \setminus \{p(s,o) \in B \mid \#_o(B) = 1\}$
$C' := C \setminus$ predicates(Keep)
$P := (A \Diamond B) \cup$ Keep $\cup \{p(\mathrm{x}_{a,b}, \mathrm{y}_{p,b}) \mid p \in C'\}$
$P :=$ Core$(P)$
return $(P, \mathrm{x}_{a,b})$
```

4.4. Kernel Explanations

As discussed in Section 4.2, when dealing with RDF-based SKBs, we can also enhance the user experience by further refining the shape of $core(\mathbf{U}, \mathcal{S})$ to hide nexus that may be considered obvious. To this aim, we are going to define the *kernel (explanation)* of $\mathbf{U}$ according to $\mathcal{S}$, denoted by $ker(\mathbf{U}, \mathcal{S})$. What we obtain is not precisely a characterization for $\mathbf{U}$ according to $\mathcal{S}$. Nonetheless, such a new formula can be understood as a special explanation that provides the user with all the "fundamental" and "expected" nexus of similarity expressed by $core(\mathbf{U}, \mathcal{S})$. To formally define $ker(\mathbf{U}, \mathcal{S})$, we exploit an ad hoc variant of $\mathcal{S}$, consisting of the $\top$-free SKB $\mathcal{S}_{\mathbf{U}} = (K, \varsigma_{\mathbf{U}})$, where $\varsigma_{\mathbf{U}}$ is defined as in Section 4.2. Note that, $\mathcal{S}_{\mathbf{U}}$ is not interesting in its own, but it has to be considered only as a technical tool for dealing with the kernel explanation of the unit $\mathbf{U}$ at hand. Interestingly, if $core(\mathbf{U}, \mathcal{S}) = x \leftarrow \top(x)$, then $ker(\mathbf{U}, \mathcal{S})$ does not exist; in all the other cases, it is not difficult to show that the set of all instances of $core(\mathbf{U}, \mathcal{S})$ and $ker(\mathbf{U}, \mathcal{S})$ according to $\mathcal{S}$ do coincide. Currently, the neXSim platform has been optimized to furnish users with kernel explanations in place of core characterizations.

4.5. Computing Kernel Explanations

We now show how to efficiently construct $ker(\mathbf{U}, \mathcal{S})$. Before analyzing the algorithm, we introduce a convenient variant of the well-known direct product operator $\otimes$, here denoted by $\Diamond$ to avoid confusion. Essentially, given two terms t and t', the product $t \Diamond t'$ is either the variable $\mathrm{x}_{t,t'}$ if $t \neq t'$ or the term t itself, otherwise. Having that, the product between structures behaves as usual [19]. For example, given $S_1 = \{\mathrm{p}(\mathrm{y}, 1)\}$ and $S_2 = \{\mathrm{p}(2, 3), \mathrm{p}(\mathrm{y}, 4), \mathrm{p}(3, 1)\}$, the new product $S_1 \Diamond S_2$ is the structure $\{\mathrm{p}(\mathrm{x}_{\mathrm{y},2}, \mathrm{x}_{1,3}), \mathrm{p}(\mathrm{y}, \mathrm{x}_{1,4}), \mathrm{p}(\mathrm{x}_{\mathrm{y},3}, 1)\}$. We are now ready to explain the behaviours of Algorithm 1 and Algorithm 2. Let us start by describing the behavior of the latter. The input of Algorithm 2 is given by two structures, namely A and B, and two terms, namely a and b. Assuming that $p(x, y) \in C$ implies that $x = a$ when $C = A$ and $x = b$ when $C = B$, our algorithm returns the pair $(\text{Core}(A \Diamond B), a \Diamond b)$, where $p(x, y) \in \text{Core}(A \Diamond B)$ implies that $x = a \Diamond b$. This assumption will always be satisfied whenever we call Algorithm 2 as a subroutine of Algorithm 1. In line 1, we introduce two functions, v_1 and v_2, which are identities outside their domains. They map a to $\mathrm{x}_{a,b}$ and b to $\mathrm{x}_{a,b}$, respectively, without altering terms common to both A and B. In line 2, we store the common predicates of A and B in the set C. This is done because elements not in the common signature are directly eliminable. In line 3, we project A and B onto their common signature, renaming both $a \in A$ and $b \in B$ as $\mathrm{x}_{a,b}$. This enables the intersection of the two sets. Note that when Algorithm 2 is used within Algorithm 1, the distinct elements $e_1, \dots, e_m$ necessitate renaming to avoid an invariably empty intersection. In line 4, we define the set *Keep*. This set comprises all atoms $p(x, y)$ common to both A and B, with y occurring exactly once in $A \cup B$. The distinctiveness of *Keep* is as follows: for any atom $p(z, z') \in B$, when $p(x, y) \Diamond p(z, z')$ is considered, with $p(x, y) \in Keep$, the term $y \Diamond z'$ appears only once in $Keep \Diamond B \subseteq A \Diamond B$. It can be mapped back to y, present in $Keep \Diamond Keep \subseteq A \Diamond B$, making the materialization of this atom superfluous. Lines 5 and 6 eliminate atoms with uniquely occurring terms from the initial structures. Line 7 stores in C' the predicates common to A and B, absent in *Keep*. Line 8 adds to the final structure atoms with fresh variables for each predicate in C', necessary due to their absence in *Keep* and their removal from A and B. *Keep* and $A \Diamond B$'s residual elements are also included. In line 9, the core of the structure is computed using an external logic program that verifies each atom's membership in the core. The element $x_{a,b}$ is not fixed because, by hypothesis, it uniquely appears in the first position of every atom, ensuring its presence in the core output of Algorithm 1. We conclude by returning the pair $(\mathrm{Core}(A \Diamond B), a \Diamond b)$. The process described in Algorithm 1 is straightforward. It involves iterating over all input summaries, as outlined in Section 4.2, and applying Algorithm 2 repeatedly.

5. Future Directions

Future work will proceed along two primary directions. The first is to increase the explanatory power of our method by extending it to capture non-local similarities, i.e., those spanning beyond the immediate neighborhood of the input entities. This is a non-trivial task that will require us to revisit the mechanisms for computing direct products and cores of structures under more relaxed assumptions. The second direction is to enhance the system's accessibility. To this end, we will focus on the automatic generation of the logical formulas into natural language, making the discovered insights truly accessible to the widest possible audience.

Acknowledgments

This work significantly contributed to the basic research activities of WP9.1: “KRR Frameworks for Green-aware AI”, supported by the FAIR project (PE_00000013, CUP H23C22000860006) – Spoke 9, within the NRRP MUR program funded by NextGenerationEU. Additional support was provided by the NRRP MUR project “Tech4You” (ECS00000009, CUP H23C22000370006) – Spoke 6, funded by NextGenerationEU; by the PRIN project PRODE “Probabilistic Declarative Process Mining” (Project ID 20224C9HXA, CUP H53D23003420006), funded by the Italian Ministry of University and Research (MUR); by the PN RIC project ASVIN “Assistente Virtuale Intelligente di Negozio” (F/360050/02/X75, CUP B29J24000200005), under the Italian National Program for Research, Innovation and Competitiveness; by the project STROKE 5.0 “Progetto, sviluppo e sperimentazione di una piattaforma tecnologica di servizi di intelligenza artificiale a supporto della gestione clinica integrata di eventi acuti di ictus” (F/310031/02/X56, CUP B29J23000430005), funded by the Italian Ministry of Enterprises and Made in Italy (MIMIT); by the NRRP project RAISE “Robotics and AI for Socio-economic Empowerment” (ECS00000035, CUP H53C24000400006), through the GOLD sub-project “Management and Optimization of Hospital Resources through Data Analysis, Logic Programming, and Digital Twin”; and by the EI-TWIN project (F/310168/05/X56, CUP B29J24000680005), funded by the former Ministry of Industrial Development (MISE, now MIMIT).

Declaration on Generative AI

During the preparation of this work, the authors used GPT-4 in order to: Grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] W. W. Cohen, A. Borgida, H. Hirsh, Computing least common subsumers in description logics, in: AAAI, AAAI Press / The MIT Press, 1992, pp. 754–760.

[2] F. Baader, R. Küsters, R. Molitor, Computing least common subsumers in description logics with existential restrictions, in: IJCAI, Morgan Kaufmann, 1999, pp. 96–103.

[3] F. Baader, B. Sertkaya, A. Turhan, Computing the least common subsumer w.r.t. a background terminology, J. Appl. Log. 5 (2007) 392–420.

[4] S. Colucci, F. M. Donini, S. Giannini, E. D. Sciascio, Defining and computing least common subsumers in RDF, J. Web Semant. 39 (2016) 62–80. URL: https://doi.org/10.1016/j.websem.2016.02.001. doi:10.1016/j.websem.2016.02.001.

[5] S. E. Hassad, F. Goasdoué, H. Jaudoin, Learning commonalities in RDF, in: ESWC (1), volume 10249 of *Lecture Notes in Computer Science*, 2017, pp. 502–517.

[6] A. Petrova, E. Sherkhonov, B. C. Grau, I. Horrocks, Entity comparison in RDF graphs, in: ISWC (1), volume 10587 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 526–541.

[7] A. Petrova, E. V. Kostylev, B. C. Grau, I. Horrocks, Query-based entity comparison in knowledge graphs revisited, in: ISWC (1), volume 11778 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 558–575.

[8] J. C. Jung, C. Lutz, F. Wolter, Least general generalizations in description logic: Verification and existence, in: AAAI, AAAI Press, 2020, pp. 2854–2861.

[9] J. C. Jung, C. Lutz, H. Pulcini, F. Wolter, Logical separability of labeled data examples under ontologies, Artif. Intell. 313 (2022) 103785. URL: https://doi.org/10.1016/j.artint.2022.103785. doi:10.1016/j.artint.2022.103785.

[10] B. ten Cate, V. Dalmau, M. Funk, C. Lutz, Extremal fitting problems for conjunctive queries, in: PODS, ACM, 2023, pp. 89–98.

[11] G. Amendola, M. Manna, A. Ricioppo, A logic-based framework for characterizing nexus of similarity within knowledge bases, Information Sciences 664 (2024) 120331. URL: https://www.sciencedirect.com/science/article/pii/S0020025524002445. doi:https://doi.org/10.1016/j.ins.2024.120331.

[12] R. Navigli, M. Bevilacqua, S. Conia, D. Montagnini, F. Cecconi, Ten years of babelnet: A survey, in: IJCAI, ijcai.org, 2021, pp. 4559–4567.

[13] G. Agresta, G. Amendola, P. Cofone, M. Manna, A. Ricioppo, Characterizing nexus of similarity between entities, in: IPS-RCRA-SPIRIT@AI*IA, volume 3585 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023.

[14] F. Baader, M. Bienvenu, C. Lutz, F. Wolter, Query and predicate emptiness in ontology-based data access, J. Artif. Intell. Res. 56 (2016) 1–59. URL: https://doi.org/10.1613/jair.4866. doi:10.1613/jair.4866.

[15] M. Calautti, S. Greco, C. Molinaro, I. Trubitsyna, Preference-based inconsistency-tolerant query answering under existential rules, Artif. Intell. 312 (2022) 103772. URL: https://doi.org/10.1016/j.artint.2022.103772. doi:10.1016/j.artint.2022.103772.

[16] J. Sandell, E. Asplund, W. Y. Ayele, M. Duneld, Performance comparison analysis of arangodb, mysql, and neo4j: An experimental study of querying connected data, in: HICSS, ScholarSpace, 2024, pp. 7760–7769.

[17] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg,

P. Selmer, A. Taylor, Cypher: An evolving query language for property graphs, in: SIGMOD Conference, ACM, 2018, pp. 1433–1445.

[18] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.

[19] B. ten Cate, V. Dalmau, The product homomorphism problem and applications, in: M. Arenas, M. Ugarte (Eds.), 18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium, volume 31 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 161–176. URL: https://doi.org/10.4230/LIPIcs.ICDT.2015.161. doi:10.4230/LIPIcs.ICDT.2015.161.