DECIDE - Detection of Contextual Identity
(Technical Report)

Joe Raad, Nathalie Pernelle, Fatiha Sais

Contents

1 Introduction

In this technical report we present $DECIDE$, a new approach for discovering contextual identity relationships in RDF knowledge bases. The problem of detecting contextual identity links can be defined as follows: given a knowledge base $\mathcal{B} = (\mathcal{O}, \mathcal{F})$ and a set $\mathcal{I}^{tc}$ of individuals of a target class tc of the ontology $\mathcal{O}$, find for the set of all individual pairs $(i_1, i_2) \in (\mathcal{I}^{tc} \times \mathcal{I}^{tc})$ the most specific contexts in which (i_1, i_2) are identical.

For instance, if we consider the example depicted in Figure 1, the two individuals $drug1$ and $drug2$ of the target class $Drug$ can be considered as identical in the context where we consider all the ontology's properties except of the property name for the drugs. On other hand, the two individuals $drug2$ and $drug3$ can be considered as identical in two most specific global contexts. The first context is the one in which we consider all the ingredients composing the drugs and for every ingredient we consider its weight. However, in this first context, the description of a weight is reduced to the measure unit without considering the quantity (property $hasValue$). A second context in which these individuals are identical is the context where we consider the weight of *Paracetamol* described by its value and its measure unit, and only the presence of *Lactose* in the drugs without considering their weight. These contexts also represent the most specific contexts in which $drug1$ and $drug3$ are identical.

We are interested in the search of the most specific global contexts involving a subset of classes, and we consider for every class a subset of properties. Some contexts can be obviously more relevant than others (e.g. a value of the weight without its measure unit does not have sense). Hence, we also aim to take into account some expert knowledge that can be represented as a set of constraints on the classes and/or properties that should or should not be involved in the considered contexts.

In the next section, we introduce the basic notions and the definitions that are needed to define a contextual identity link, which is based on the notion of global contexts. In section 3, we present our approach of detecting contextual identity links. Finally, in section 4 we present the output of $DECIDE$ on some examples, with each example presenting a specific case.

2 Preliminaries

Before we present the algorithm in 3, we introduce in this section the terminologies that are used throughout the algorithm.

2.1 Knowledge Base

We consider a knowledge base where the ontology is represented in OWL[1] and the data represented in RDF[2]. A knowledge base $\mathcal{B}$ is defined by a couple $(\mathcal{O}, \mathcal{F})$ where:
– the ontology $\mathcal{O} = (\mathcal{C}, \mathcal{DP}, \mathcal{OP}, \mathcal{A})$ is defined by a set of classes $\mathcal{C}$, a set of *owl:DataTypeProperty* $\mathcal{DP}$, a set of *owl:ObjectProperty* $\mathcal{OP}$, and a set of axioms $\mathcal{A}$ (e.g property domains or ranges, subsumption).

– $\mathcal{F}$ is a collection of triples *(subject, property, object)*, that expresses that some relationship, indicated by the property, holds between the subject and object of the triple (between two resources or between a resource and a literal value) [3].

[1] https://www.w3.org/OWL/
[2] https://www.w3.org/RDF/
[3] We do not consider blank nodes in this work

2.2 Contexts

The contextual identity link detection is based on identifying the most specific global contexts where two instances are identical. We consider a global context as a connected sub-ontology of $\mathcal{O}$ which represents the vocabulary on which two instances are considered as identical. We first introduce the set of classes $DepC$ that can be involved in the contexts. Then, we formally define the global contexts and the contextual identity relation, named $identiConTo$, that expresses that two instances are identical in a given global context.

A global context is represented as a subset of classes and properties of the ontology, and a set of axioms which is limited to constraints on property domains and ranges. We automatically choose the abstraction level of the classes involved in a global context by selecting, from the instantiated classes (direct types), the most general ones.

Definition 2.1. Selected classes $DepC$. The set of selected classes $DepC$ that can be involved in the contextual identity links is the subset of instantiated classes c_i of $\mathcal{B}$ such that:

$$DepC = \{c_i \in \mathcal{C} \mid \nexists c_j \in \mathcal{C} \ s.t. \ \exists x, directType(x, c_j) \ and \ c_i \sqsubset c_j\}$$

Example 1. In Figure 1, $DepC$ will contain all the classes of the graph except of $Product$ which is not instantiated. Therefore, $par1$ and $lac1$ will be uniquely considered as of type $Paracetamol$ and $Lactose$ respectively.

Definition 2.2. Global Context. A global context is a sub-ontology GC_u=(C_u, DP_u, OP_u, A_u) of $\mathcal{O}$ such that $C_u \subseteq DepC$, $DP_u \subseteq DP$, $OP_u \subseteq OP$ and A_u is a set of domain and range constraints that are more specific than those described in A: $\forall op \in OP_u$, $domain_u(op) \sqsubseteq domain_{\mathcal{O}}(op)$ and $range_u(op) \sqsubseteq range_{\mathcal{O}}(op)$, and $\forall dp \in DP_u$, $domain_u(dp) \sqsubseteq domain_{\mathcal{O}}(dp)$.

Example 2. In Figure 1, there exists many possible global contexts. We present one:
GC_1=$(C = \{Drug, Paracetamol, Lactose, Weight\}$,
$OP = \{isComposedOf, \ hasWeight\}$, $DP = \{hasValue\}$,
$A = \{domain(isComposedOf) = Drug$,
$range(isComposedOf) = Lactose \sqcup Paracetamol$,
$domain(hasWeight) = Lactose \sqcup Paracetamol$,
$range(hasWeight) = Weight\})$

Definition 2.3. Order relation between global contexts. Let $GC_u = (C_u, OP_u, DP_u, A_u)$ and $GC_v = (C_v, OP_v, DP_v, A_v)$ be two global contexts. The context GC_u is more specific than GC_v, noted $GC_u \leq GC_v$, if $C_v \subseteq C_u$, $OP_v \subseteq OP_u$, $DP_v \subseteq DP_u$ and
$\forall p \in OP_v \cup DP_v$, $domain_u(p) \sqsubseteq domain_v(p)$, and $\forall p \in OP_v$, $range_v(p) \sqsubseteq range_u(p)$.

In order to filter out the irrelevant contexts to consider, we take in consideration the experts' knowledge when it is available. An expert can supply three types of constraints:

– *Unwanted properties* (UP): this refers to properties that an expert wants to discard in the detection of contextual identity links. Such constraints can be used when property values correspond to unstructured (free) text, or are known to be particularly heterogeneous, or when the property subjects or objects are evolutive or insignificant to compare two instances for a given task. In such cases, an expert can declare that a property p is unwanted for a given domain c_i (or a particular range c_j) by adding a constraint $up = (c_i, p, *)$ (resp. $up = (*, p, c_j)$) in UP. When a property is unwanted in all domains and ranges, the constraint $(*, p, *)$ can be used. In such cases, $p \notin OP \cup DP$.

– *Necessary properties* (NP): a necessary property is a constraint noted $np = (c_i, p, *)$ or $(*, p, c_j)$. When such constraints are added to NP, we will only consider global contexts where the property $p \in OP$ or $p \in DP$, and such that $c_i \in domain(p)$ (resp. $c_j \in range(p)$).

– *Co-occurring properties* (CP): a co-occurrence constraint $cp = \{(c_i, p_1, *)$, ..., $(c_i, p_n, *)\}$ can be declared to guarantee that a certain class c_i will be either declared as the domain (or range) of *all* the properties indicated in the constraint, or *none* of them. For instance, to declare that the weight's value has no meaning without its measure unit, an expert can add the constraint $cp_1 = \{(Weight, hasValue, *)$, $(Weight, hasUnit, *)\}$.

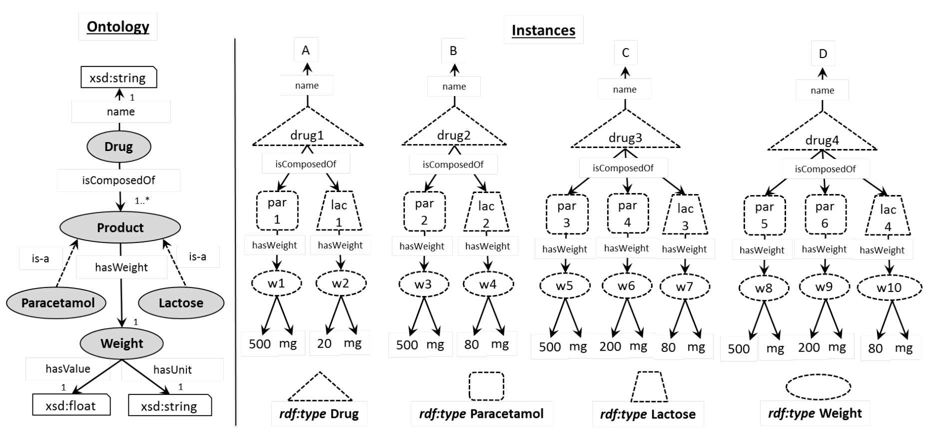

Figure 1: An extract of ontology $\mathcal{O}$, four individuals $drug1$, $drug2$, $drug3$, and $drug4$ of the target class $Drug$.

Definition 2.4. Local Contexts. A local context of a class c is a context that is limited to datatype and object properties that are defined for c. In the algorithm, we will note:
– LC_u^{out}=$(C_u^{out}, OP_u^{out}, DP_u^{out}, A_u^{out})$, a local context where $\forall p \in OP_u^{out} \cup$

DP_u^{out}, $domain_u^{out}(p)$=c and
–LC_u^{in}=(C_u^{in}, OP_u^{in}, DP_u^{in}, A_u^{in}) a local context where DP_u^{in}=$\emptyset$ and $\forall op \in OP_u^{in}$, $range_u^{in}(op)$=c.

2.3 Contextual identity links

In our approach, two instances are considered identical in a given context, when all the information described in the context is known and represented in the instances' descriptions, and when these descriptions are equal. Therefore, we firstly define the contextual description that is considered for one instance in one context. Then we will define the conditions that must hold to consider that two RDF descriptions refer to identical instances in a given context.

Definition 2.5. Contextual instance description according to a global context. Given a RDF dataset $\mathcal{F}$, a global context $GC_u = (C_u, OP_u, DP_u, A_u)$ and an instance i, a contextual description G_i of i in GC_u is the maximal set of triples that describe i in $\mathcal{F}$ such that:
– G_i forms a connected graph that contains at least one triple where i is subject or object
– $\forall\ t = <s,\ p,\ o> \in G_i$ then $p \in OP_u \cup DP_u$ and $type(s) \sqsubseteq domain_u(p)$ and $type(o) \sqsubseteq range_u(p)$
– $\forall\ j$ a class instance of G_i, and $\forall dp \in DP_u$ such as $type(j) \sqsubseteq domain(dp)$, then $\exists\ t_a = <j,\ p,\ v> \in G_i$, with v of type literal
– $\forall\ j$ a class instance of G_i, and $\forall op \in OP_u$ such as $type(j) \sqsubseteq domain(op)$, and $c_1 \cup c_2 \sqsubseteq range(op)$ then $\exists\ t_a = <j,\ op,\ k>$ and $t_b = <j,\ op,\ l> \in G_i$ with $type(k) = c_1$ and $type(l) = c_2$

From two contextual descriptions of two class instances defined in a given context, we can define if they can be considered as identical. In this work we will consider that properties are local complete: if a property p is instantiated for a given class instance i, we consider that all the property instances are known for i. Since a local completness is assumed, two instances can be considered as identical when the contextual graphs, formed by the contextual descriptions, are isomorphic up to a renaming of the instance URI. Note that since some classes can be removed from the global context, this constraint can in fact be considered class by class.

Definition 2.6. Identity in a global context. Given a global context GC_u, a pair of instances (i_1, i_2) are identical in GC_u, noted $identiConTo_{<GC_u>}(i_1, i_2)$, only if the two labelled graphs G_{i_1} and G_{i_2} that represent the contextual descriptions of i_1 and i_2 are isomorphic up to a rewriting of the URI of the class instances (literals must be equal).

Example 3. $drug1$ and $drug2$ are considered as identical according to the global context GC_1 defined in Example 2.
(i.e. $identiConTo_{<GC_1>}(drug1, drug2)$).

The contextual identity relations will only be specified for the most specific global context(s), but can be inferred for the more general ones using the order relation between global contexts: given GC_u and GC_v two global contexts, with $GC_u \leqslant GC_v$, then $identiConTo_{<GC_u>}(i_1, i_2) \Rightarrow identiConTo_{<GC_v>}(i_1, i_2)$.

2.4 Identity Graph

An identity graph $IG_{<i_1,i_2>}$=(V, E) for a pair of individuals (i_1, i_2), is a connected labelled directed graph, where V is a set of nodes and E is a set of edges. Each node n_i represents a set of pairs $I_1 \times I_2$, and the local contexts $LC_n^{in}(c)$ and $LC_n^{out}(c)$ that generalize all the most specific local contexts $LC_n^{in}(c)$ and $LC_n^{out}(c)$ for which the pairs are considered as identical. A node n_1 representing a set of pairs $I_1 \times I_2$ is linked to a node n_2 representing the set of pairs $J_1 \times J_2$ by an edge $e(n1, n2)$ labelled as p, if $\forall$ (i_1, i_2) $\in I_1 \times I_2$, $\exists$ $j_1 \in J_1$ and $j_2 \in J_2$ such that:
– $\exists < i_1, p, j_1 >$ and $< i_2, p, j_2 > \in \mathcal{F}$ if $p \in LC_{n_1}^{out}(c)$
– $\exists < j_1, p, i_1 >$ and $< j_2, p, i_2 > \in \mathcal{F}$ if $p \in LC_{n_1}^{in}(c)$.
In an identity graph $IG_{<i_1,i_2>}$, a graph path gp_i is a sequence of distinct nodes $\{n_1, n_2, ..., n_m\}$ rooted by n_1 that represents (i_1, i_2) and every graph paths are such that: $\nexists$ n_k, $n_l \in gp_i$, with $k < l$ and $LC_{n_l}(c) \leq LC_{n_k}(c)$.

Figure 2 presents the identity graphs IG_1 and IG_2 of the pair of drugs $(drug3, drug4)$.

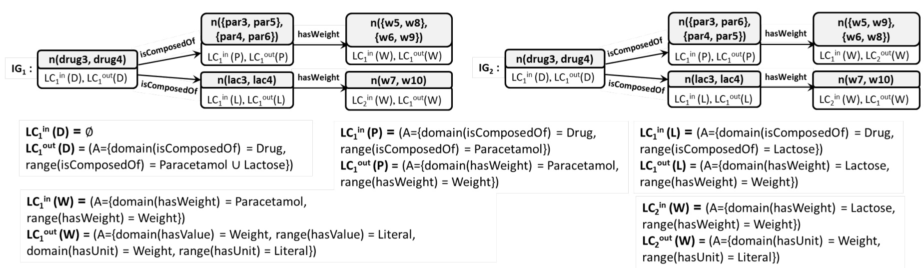

Figure 2: The two possible Identity Graphs for the pair (drug3, drug4). For simplicity reasons, C, OP, and DP are not represented in this Figure for all local contexts.

3 Algorithm

The goal of the algorithm $DECIDE$ (DEtection of Contextual IDEntity) is to determine for each pair of individuals (i_1, i_2) $\in I^{tc} \times I^{tc}$ of a target class tc given by the user, the set of the most specific global contexts in which the identity relation $identiConTo$ is true. $DECIDE$ requires to have the set of facts $\mathcal{F}$ of the considered knowledge base, the target class tc as inputs, and may consider different constraint lists UP, NP, CP given by an expert. The algorithm $DECIDE$, described in Algorithm 1, is composed of three main steps:

Algorithm 1: DECIDE

Input:
– tc: the target class
– $K(NP, UP, CP)$: the expert constraints
– $\mathcal{F}$: the set of facts of the considered knowledge base

Output: $MScontexts$: set of most specific global contexts for each pair

$DepC \leftarrow getDepC(\mathcal{F})$;
$I^{tc} \leftarrow$ list of instances of $type(tc)$ in $\mathcal{F}$;
foreach ($(i_1, i_2) \in I^{tc} \times I^{tc}$) **do**
    $GCset \leftarrow \emptyset$;
    $IGset \leftarrow constructIdentityGraph(i_1, i_2, DepC, K, \mathcal{F})$;
    foreach ($IG \in IGset$) **do**
        $n_0 \leftarrow IG.getNode(i_1, i_2)$;
        $N \leftarrow \emptyset$; $a \leftarrow \emptyset$; $GC \leftarrow \emptyset$; $LCset \leftarrow \emptyset$;
        $GC \leftarrow generateGC(n_0, a, GC, LCset, N, IG)$;
        $GCset.add(GC)$;
        foreach ($LC \in LCset$) **do**
            $GC \leftarrow \emptyset$; $GC.add(LC)$;
            $GC \leftarrow generateGC(n_0, a, GC, LCset, N, IG)$;
            if ($\nexists\ GC_1 \in GC_{set},\ such\ as\ GC_1 \leq GC$) **then**
                $GCset.add(GC)$;
            if ($\exists\ GC_2 \in GC_{set},\ such\ as\ GC \leq GC_2$) **then**
                $GCset.remove(GC_2)$;
    $MScontexts.add(GCSet, (i_1, i_2))$;
return $MScontexts$;

3.1 Collect the Selected classes

$DECIDE$ starts by collecting the selected classes (definition **??**), in order to indicate the level of abstraction to be considered in building the identity graphs and generating the most specific global contexts. The collection of these classes is described in Algorithm 2.

Then for each pair of individuals of the target class tc, $DECIDE$ proceeds with the two following steps:

Algorithm 2: get DepC

```
Input: F: the set of facts of the considered knowledge base
Output: DepC: the set of classes that can be involved in the contextual
        identity links
DepC ← ∅ ;
foreach (s, rdf:type, o) ∈ F do
    if DepC isEmpty then
        DepC.add(o);
    else
        unwantedClasses ← ∅ ;
        foreach c ∈ DepC do
            if (rdfs:subClassOf(c, o) then
                unwantedClasses.add(c);
            if (rdfs:subClassOf(o, c) then
                unwantedClasses.add(o);
        DepC.add(o);
        DepC.removeAll(unwantedClasses);
return DepC;
```

3.2 Construct the Identity Graph

DECIDE constructs for each pair of individuals of *tc*, the identity graph using a depth-first search algorithm. When different mappings between instances of the same class can be considered, a new identity graph identity is constructed. The construction of the identity graph(s) is described in Algorithm 4.

3.3 Generate the most specific global context

The generation of the most specific global contexts is based on the constructed identity graphs. A global context GC is constructed using the set of local contexts and insures the presence of no more than one local context per class in each global context. The most specific global contexts are generated using the function *generateGC*, which traverses the identity graph IG using also a depth-first search algorithm. This function, described in Algorithm 5, aims to add its most specific outgoing local context $LC_n(c)$, which is already calculated in IG, to current global context GC (i.e. the most specific global context). There is three cases:

(1) If GC does not contain a local context $LC_{ex}(c)$ for the class c, or if GC contains $LC_{ex}(c)$ with $LC_{ex}(c)$ equal to the local context $LC_n(c)$ of n, then $LC_n(c)$ is added to GC. This function is then recursively recalled for each node

Algorithm 3: construct Identity Graph

Input:
– tc: the target class
– (i_1, i_2): pair of individuals of the class tc
– $depC$: the set of classes collected in the preprocessing
– $\mathcal{F}$: the set of facts of the considered knowledge base
Output: $IGset$: set of the possible identity graphs for (i_1, i_2)

$IG_1 \leftarrow \emptyset$; $LCset \leftarrow \emptyset$; $IGset \leftarrow \emptyset$;
$n(i_1, i_2) \leftarrow emptyGraphNodeForIndivPair(i_1,\ i_2)$;
$IGset.add(IG_1)$;
foreach $IG \in IGset$ **do**
    $IG \leftarrow ExpandIdentityGraph(n(i_1, i_2), LCset, IG, IGset, DB, depC, \mathcal{F})$;
return $IGset$;

n_d in IG, such as there is an edge from n to n_d.

(2) If GC contains a local context $LC_{ex}(c)$ for the class c, and $LC_n(c)$ is more specific than $LC_{ex}(c)$, then this function is recursively recalled for each destination node n_d in IG, such as there is an edge from n to n_d labelled op and we have in the axioms of $LC_{ex}(c)$: domain of $op = c$ and $type(n_d) \sqsubseteq range(op)$.

(3) If GC contains a local context $LC_{ex}(c)$ for the class c, and $LC_n(c)$ is not more specific than $LC_{ex}(c)$, then this function is not recalled for this graph node, and the domain representing the type of the node source and the range representing c of the object property op that led to this graph element will be removed from $LC_{ex}(c)$.

In both (2) and (3), $LC_n(c)$ and the most specific local context that generalizes $LC_n(c)$ and $LC_{ex}(n)$ will be added to a list $LCset$, in order to guarantee the presence of these local contexts in other global contexts. Therefore, resulting in several most specific global contexts for the same pair.

The time complexity of this algorithm is $O(n \times I^2)$, with $n =$ the number of pairs of the target class tc, and $I =$ the number of instances in $\mathcal{F}$. $DECIDE$ is implemented in $Java$ using the $Jena\ TDB$ triple store, and is available at `http://github.com/raadjoe/DECIDE_v2`.

When applied on the pair $(drug1, drug2)$, $DECIDE$ will result in two most specific contexts GC_1 and GC_2, representing the most specific global contexts in which these two drugs are identical:

GC_1=($C = \{Drug, Paracetamol, Lactose, Weight\}$,
$OP = \{isComposedOf,\ hasWeight\}$, $DP = \{hasUnit\}$,
$A = \{domain(isComposedOf) = Drug$,

Algorithm 4: ExpandIdentityGraph

Input:
– $n(i_1, i_2)$: a graph node of a pair of individuals
– IG: the identity graph
– $IGset$: set of the possible identity graphs for (i_1, i_2)
– $depC$: the selected classes
– $\mathcal{F}$: the set of facts of the considered knowledge base

Output: IG: the expanded identity graph

$c \leftarrow getDepClass((i_1, i_2), depC)$;
$lc_{max}(c) \leftarrow getMostSpecificLocalContext(i_1, i_2, \mathcal{F})$;
if $\exists\ lc(c) \in LCset,\ with\ lc(c) \nleq lc_{max}(c)$ **then**
    $lc_{max} \leftarrow getHighestCommonLocalContext(lc_{max}(c), lc(c))$;
$LCset.add(lc_{max})$;
$n(i_1, i_2).add(lc_{max})$;
$IG.add(n(i_1, i_2))$;
foreach $(p \in lc_{max})$ **do**
    if $rdf{:}type(p, owl : ObjectProperty)$ **then**
        $vals_{i_1} \leftarrow getValuesWithSameDepC(p, i_1, \mathcal{F})$;
        $vals_{i_2} \leftarrow getValuesWithSameDepC(p, i_2, \mathcal{F})$;
        $CMB \leftarrow getAllNodesCombinations(vals_{i_1}, vals_{i_2})$;
        $IGset' \leftarrow \emptyset$;
        foreach $(cmb \in CMB)$ **do**
            $n(cmb) \leftarrow emptyGraphNodeForAllPairs(j_1,\ j_2) \in cmb$;
            foreach $(IG \in IGset)$ **do**
                $IG' \leftarrow IG$;
                $IG'.add(n(cmb))$;
                $IGset'.add(IG')$;
            $IGset' \leftarrow ExpandIdentityGraph(n, IG', IGset', depC, \mathcal{F})$;
return $IGset$;

Algorithm 5: Generate GC

Input:

– n: an identity graph node
– a_s: axiom indicating the type of the node source with the property source
– GC: the current global context
– $LCset$: set of unused local contexts
– N: list of visited nodes
– IG: the identity graph

Output: GC: the current most specific global context

if $(n \notin N)$ **then**
    $N.add(n)$;
    $LC_n(c) \leftarrow getOutgoingLocalContext(n)$;
    $LC_{ex}(c) \leftarrow GC.getExistingLocalContext(c)$;
    if $(LC_{ex}(c) == null\ or\ LC_{ex}(c) == LC_n(c))$ **then**
        $GC.add(LC_n(c))$; //if it does not exist
        $E^n \leftarrow IG.getOutgoingEdges(n)$;
        foreach $(e = (op, n, n_d) \in E^n)$ **do**
            $a_d \leftarrow \{domain(op) = c, range(op) = type(n_d)\}$;
            $GC \leftarrow generateGC(n_d, a_d, GC, LCset, N, IG)$;
    else
        if $(LC_n(c) \leqslant LC_{ex}(c))$ **then**
            $E^n \leftarrow IG.getOutgoingArcs(n)$;
            foreach $(e = (op, n, n_d) \in E^n)$ **do**
                $a_d \leftarrow \{domain(op) = c, range(op) = type(n_d)\}$;
                if $(a_d \in LC_{ex}(c))$ **then**
                    $GC \leftarrow generateGC(n_d, a_d, GC, LCset, N, IG)$;
        else
            $c_s \leftarrow a_s.getDomain()$;
            $LC(c_s) \leftarrow GC.getExistingLocalContext(c_s)$;
            $LC(c_s).remove(a_s)$;
            $GC.replace(LC(c_s))$; //replace existing $LC(c_s)$
        $LCset.add(LC_n(c))$; //if it does not exist
        $LCset.intersect(LC_n(c), LC_{ex}(c))$; //if it does not exist
return GC;

$range(isComposedOf) = Lactose \sqcup Paracetamol$,
$domain(hasWeight) = Lactose \sqcup Paracetamol$,
$range(hasWeight) = Weight$,
$domain(hasUnit) = Weight$, $range(hasUnit) = xsd{:}string\})$

GC_2=($C = \{Drug, Paracetamol, Lactose, Weight\}$,
$OP = \{isComposedOf,\ hasWeight\}$, $DP = \{hasValue, hasUnit\}$,
$A = \{domain(isComposedOf) = Drug$,
$range(isComposedOf) = Lactose \sqcup Paracetamol$,
$domain(hasWeight) = Paracetamol$,
$range(hasWeight) = Weight$,
$domain(hasValue) = Weight$, $range(hasValue) = xsd{:}float$,
$domain(hasUnit) = Weight$, $range(hasUnit) = xsd{:}string\})$

4 Example

In this section, we present the output of $DECIDE$ with different examples, with each one presenting a specific case such as:

- Multi-valued properties, with the properties' objects having different types
- Multi-valued properties, with the properties' objects having the same type
- Different properties with the same object

4.1 Multi-valued properties, with the properties' objects having different types

In this section, we present the outcome of $DECICE$ in the case of multi-valued properties, with the properties' objects having different types. Figure 3 presents three drugs, with each drug is composed of one instance of type $Paracetamol$ and one instance of type $Lactose$.
Input: $Example1.rdf$, $Drug$
Output: $identiConTo_{<GC_1(Drug)>}(drug1, drug2)$,
$identiConTo_{<GC_2(Drug)>}(drug1, drug3)$, $identiConTo_{<GC_3(Drug)>}(drug1, drug3)$,
$identiConTo_{<GC_2(Drug)>}(drug2, drug3)$, $identiConTo_{<GC_3(Drug)>}(drug2, drug3)$,

with GC_1=($C = \{Drug, Paracetamol, Lactose, Weight\}$,
$OP = \{isComposedOf,\ hasWeight\}$, $DP = \{hasValue, hasUnit\}$,
$A = \{domain(isComposedOf) = Drug$,
$range(isComposedOf) = Lactose \sqcup Paracetamol$,
$domain(hasWeight) = Lactose \sqcup Paracetamol$,
$range(hasWeight) = Weight$,
$domain(hasValue) = Weight$,
$range(hasValue) = Literal$,
$domain(hasUnit) = Weight$,

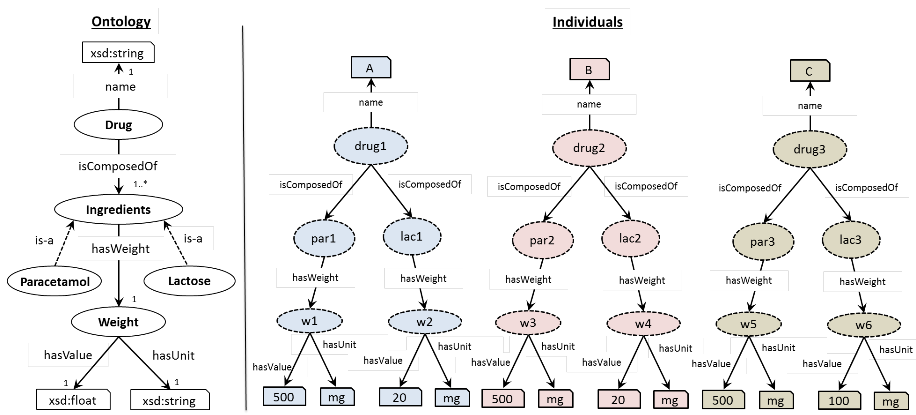

Figure 3: Example 1

$range(hasUnit) = Literal\})$

and GC_2=$(C = \{Drug, Paracetamol, Lactose, Weight\}$,
$OP = \{isComposedOf,\ hasWeight\},\ DP = \{hasValue, hasUnit\}$,
$A = \{domain(isComposedOf) = Drug$,
$range(isComposedOf) = Lactose \sqcup Paracetamol$,
$domain(hasWeight) = Paracetamol$,
$range(hasWeight) = Weight$,
$domain(hasValue) = Weight$,
$range(hasValue) = Literal$,
$domain(hasUnit) = Weight$,
$range(hasUnit) = Literal\})$

and GC_3=$(C = \{Drug, Paracetamol, Lactose, Weight\}$,
$OP = \{isComposedOf,\ hasWeight\},\ DP = \{hasUnit\}$,
$A = \{domain(isComposedOf) = Drug$,
$range(isComposedOf) = Lactose \sqcup Paracetamol$,
$domain(hasWeight) = Lactose \sqcup Paracetamol$,
$range(hasWeight) = Weight$,
$domain(hasUnit) = Weight$,
$range(hasUnit) = Literal\})$

4.2 Multi-valued properties, with the properties' objects having the same type

In this section, we present the outcome of $DECICE$ in the case of multi-valued properties, with the properties' objects having the same type. Figure 4 presents three drugs with the same ontology of Figure 3. However in this example, $drug1$

and $drug2$ are composed of two instances of type $Paracetamol$ and one instance of type $Lactose$, while $drug3$ is composed of two instances of type $Paracetamol$ and two instances of type $Lactose$.

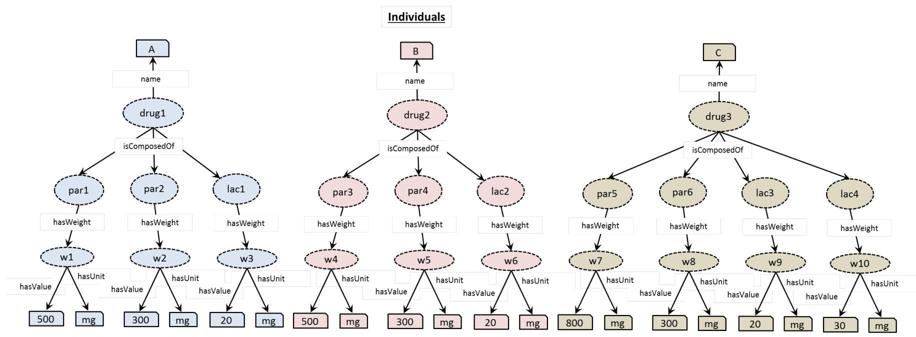

Figure 4: Example 2

Input: $Example2.rdf$, $Drug$
Output: $identiConTo_{<GC_4(Drug)>}(drug1, drug2)$,
$identiConTo_{<GC_5(Drug)>}(drug1, drug3)$,
$identiConTo_{<GC_5(Drug)>}(drug2, drug3)$

with GC_4=($C = \{Drug, Paracetamol, Lactose, Weight\}$,
$OP = \{isComposedOf,\ hasWeight\}$, $DP = \{hasValue, hasUnit\}$,
$A = \{domain(isComposedOf) = Drug$,
$range(isComposedOf) = Lactose \sqcup Paracetamol$,
$domain(hasWeight) = Lactose \sqcup Paracetamol$,
$range(hasWeight) = Weight$,
$domain(hasValue) = Weight$,
$range(hasValue) = Literal$,
$domain(hasUnit) = Weight$,
$range(hasUnit) = Literal\}$)

and GC_5=($C = \{Drug, Paracetamol, Weight\}$,
$OP = \{isComposedOf,\ hasWeight\}$, $DP = \{hasUnit\}$,
$A = \{domain(isComposedOf) = Drug$,
$range(isComposedOf) = Paracetamol$,
$domain(hasWeight) = Paracetamol$,
$range(hasWeight) = Weight$,
$domain(hasUnit) = Weight$,
$range(hasUnit) = Literal\}$)

4.3 Different properties with the same object

In this section, we present the outcome of $DECICE$ in the case of different properties having the same object. Figure 5 presents three drugs with their corresponding companies.

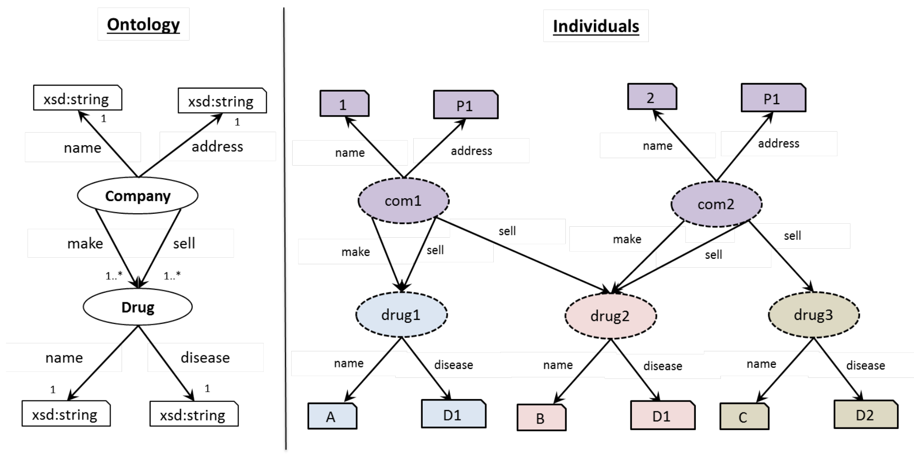

Figure 5: Example 3

Input: $Example3.rdf$, $Drug$
Output: $identiConTo_{<GC_6(Drug)>}(drug1, drug2)$,
$identiConTo_{<GC_7(Drug)>}(drug1, drug3)$,
$identiConTo_{<GC_8(Drug)>}(drug2, drug3)$

with GC_6=($C = \{Drug, Company\}$,
$OP = \{make\}$, $DP = \{address, disease\}$,
$A = \{domain(make) = Company$,
$range(make) = Drug$,
$domain(address) = Company$,
$range(address) = Literal$,
$domain(disease) = Drug$,
$range(disease) = Literal\}$)

and GC_7=($C = \{Drug, Company\}$,
$OP = \{sell\}$, $DP = \{address\}$,
$A = \{domain(sell) = Company$,
$range(sell) = Drug$,
$domain(address) = Company$,
$range(address) = Literal\}$)

and GC_8=($C = \{Drug\}$, $OP = \{\emptyset\}$, $DP = \{\emptyset\}$, $A = \{\emptyset\}$)