Computing Minimal Definition Signatures in Description Logic Ontologies

David Geleta Terry R. Payne Valentina Tamma

Abstract

The ability to rewrite defined ontological entities into syntactically different, but semantically equivalent forms is an important property of *Definability*. While rewriting has been extensively studied, the practical applicability of currently existing methods is limited, as they are bounded to particular Description Logics (DLs), and they often present only theoretical results. Moreover, these efforts focus on computing single definitions, whereas the ability to find the complete set of alternatives, or even just their signature, can support ontology alignment, and semantic interoperability in general. As the number of possible rewritings is potentially exponential in the size of the ontology, we present a novel approach that provides a comprehensive and efficient way to compute in practice all definition signatures of most defined entities described using a DL language for which the a particular definability property holds (*Beth definability*). In this paper we present our approach, and we assess the prevalence, extent and merits of definability over large and diverse corpora, as well as discussing ways in which it can be exploited to support a variety of different ontology engineering tasks.

1 Introduction

The ability to rewrite defined ontological entities into syntactically different, but semantically equivalent forms is an important property of the notion of *Definability*. In particular, *Beth definability* [2, 11] is a well-known property from classical logic, that relates the notion of *implicit definability* to the one of *explicit definability*, by stating that every implicitly defined concept is also explicitly definable, in any definitorially complete DL language [26]. For example, given an ontology $\mathcal{O} = \{\mathsf{C} \equiv \mathsf{A} \sqcup \mathsf{B}, \mathsf{A} \sqsubseteq \neg\mathsf{B}, \mathsf{D} \sqsubseteq \exists \mathsf{r}.\top\}$, where the concept C is defined explicitly, i.e. $\mathsf{C} \equiv \mathsf{A} \sqcup \mathsf{B}$, the concept A is defined implicitly under $\mathcal{O}$ by the set of general concept inclusions $\{\mathsf{C} \equiv \mathsf{A} \sqcup \mathsf{B}, \mathsf{A} \sqsubseteq \neg\mathsf{B}\}$. Thus, A can be explicitly defined by the axiom $\mathsf{A} \equiv \mathsf{C} \sqcap \neg\mathsf{B}$.

Definability in general (and Beth definability in particular) have been utilised within Description Logics (DL) to generate syntactically different, albeit semantically equivalent definitions. Known as *rewriting*, this process is primarily used for: 1) extracting equivalent terminology from a general TBox [1]; and 2) finding equivalent query rewritings in ontology-based data access scenarios [21].

These approaches exploit the fact that any defined concept has one or more possible alternative definitions; however they usually focus on finding a single alternate definition; whereas several ontology engineering tasks would benefit from the ability to identify a complete set. For example, in *ontology alignment* [7], several approaches have been proposed that successfully align ontologies [3]. However, Stuckenschmidt et. al. have argued that existing approaches often fail to compute complex correspondences: typically, systems are only able to identify simple equivalence statements between class or relation names, but often fail to identify richer semantic relation between elements of different ontologies [23]. Thus, the ability to rewrite concept definitions can widen the search space for possible correspondences. This is illustrated by the fact that some alignment mechanisms may not find a simple correspondence for some concept C, but given its definition $\mathsf{C} \equiv \mathsf{A} \sqcup \mathsf{B}$, finding a complex correspondence may be possible.

Determining the complete set of possible definitions of defined concepts is a challenging task, as the number of different definitions is potentially exponential in the size of the ontology. This is problematic for large scale ontologies, such as SNOMED CT[1] or the Foundational Model of Anatomy (FMA)[2]. Existing rewriting algorithms are *language dependent*, and thus different approaches to construct rewritings are used for different types of DL expressivity. Furthermore, even if there was an existing approach for a given language, many rewriting systems provide only a theoretical characterization of the rewriting mechanism, therefore making them less usable in practice. Finally, rewriting *requires a seed signature* to be specified in input, i.e. a restricted vocabulary to be used in defining a given concept. The process of identifying all valid seed signatures is inherently complex, as it requires examining each member of the powerset of the ontology signature and verifying whether it actually implicitly defines a particular entity. Therefore, reducing the search space for these problems is highly desirable.

In this paper we present a pragmatic approach to computing the complete set of rewriting signatures for a given ontology. Our approach exploits the Beth definability property to identify all possible alternative definitions of defined entities. We present the notion of Beth definability in Section 2, and then introduce our novel approach, that *in practice*, can efficiently compute the complete set of definition signatures (DS) of defined entities, for any DL language, where the Beth property holds (Section 3). Section 5 presents *concept definition patterns* (CDP), that not only aid comprehension of definability, but also serve as input for a *heuristic-based rewriting approach*, which produces definition axioms without using reasoning services. As little is currently known about the prevalence and the extent of definability in real world ontologies, the practical value of the various applications of definability is explored in practice through comprehensive evaluation over a large and diverse ontology corpus containing a wide range of ontologies, which illustrates the *definability landscape*, and paves the way for further research into its application areas.

[1] `http://www.ihtsdo.org/snomed-ct`
[2] `http://si.washington.edu/projects/fma`

2 Beth Definability in Description Logics

The **vocabulary** of a DL ontology[3] consists of the (disjoint) union of the countably infinite sets of concept names ($\mathsf{N}_\mathcal{C}$), role names ($\mathsf{N}_\mathcal{R}$) and individual names ($\mathsf{N}_\mathcal{I}$), where an **entity** e is either a concept or role.

A **signature** is an arbitrary set of role and concept names, and individuals; by $\mathsf{Sig}(\mathsf{C})$ we denote the signature of the a complex concept C, while $\mathsf{Sig}(\mathcal{T})$ denotes the signature of a TBox. In this paper, Σ refers to a *definition signature* (DS), i.e. the set of concept and role names that implicitly define a given concept.

A DS is used to characterise *implicitly definable* concepts in terms of their explicit definability, by exploiting *Beth definability theorem*. The theorem, initially studied for first-order logic [2], states that a concept is *implicitly definable* with respect to a theory if and only if it is also *explicitly definable*. Given that explicit definability implies implicit definability, the Beth definability property holds for some logic language $\mathcal{L}$ if the converse also holds, i.e. if implicit definability implies explicit definability. Consequently, if a term is implicitly defined then it is always possible to define it explicitly. As there are several variants of Beth definability [26], we focus on *Projective Beth definability* which is a stronger formulation [11] with the ability to specify a set of predicates Σ, thus permitting us to restrict the vocabulary that can be used in definitions. Beth definability has also been studied in the context of DLs [26], where it has been used to compute explicit definitions based on implicit definitions. We thus assume a general DL language $\mathcal{L}$ for which the Beth definability property holds. We define an explicit definability concept as:

Definition 1 ***Explicitly defined concept*** *Let C be a concept name, and $\mathcal{T}$ a TBox, where $\mathsf{C} \in \mathsf{Sig}(\mathcal{T})$. C is explicitly defined under $\mathcal{T}$, if and only if there is an axiom $\alpha : \mathsf{C} \equiv \mathsf{D}$, such that $\alpha \in \mathcal{T}$, where D is either a concept name in $\mathcal{T}$, or a complex concept such that $\mathsf{Sig}(\mathsf{D}) \subseteq \mathsf{Sig}(\mathcal{T}) \setminus \{\mathsf{C}\}$.*

For example, let us consider $\mathcal{T}^{Family}$, a small $\mathcal{ALC}$-TBox describing the family domain, shown in Figure 1 (upper). The concept Parent, defined by the axioms α_1 and α_2, is the only explicitly defined concept in the ontology. Similarly, we can define implicitly definable concepts:

Definition 2 ***Implicitly definable concept*** *Let C be a concept name, $\mathcal{T}$ a TBox, and Σ a signature, where $\mathsf{C} \in \mathsf{Sig}(\mathcal{T})$, and $\Sigma \subseteq \mathsf{Sig}(\mathcal{T}) \setminus \{\mathsf{C}\}$. C is implicitly definable from Σ under $\mathcal{T}$, if and only if for any two models $\mathcal{I}$ and $\mathcal{K}$ of $\mathcal{T}$, $\Delta^\mathcal{I} = \Delta^\mathcal{K}$, and for all predicate $P \in \Sigma$, $P^\mathcal{I} = P^\mathcal{K}$. Then it holds that $\mathsf{C}^\mathcal{I} = \mathsf{C}^\mathcal{K}$.*

Given the example, it can be seen that both Mother and Father are implicitly defined concepts in $\mathcal{T}^{Family}$, and each has six syntactically different, but semantically equivalent definitions (Figure 1, lower).

[3] In this paper, we assume familiarity with basic notions of Description Logics [1] and the Web Ontology Language [10] (OWL).

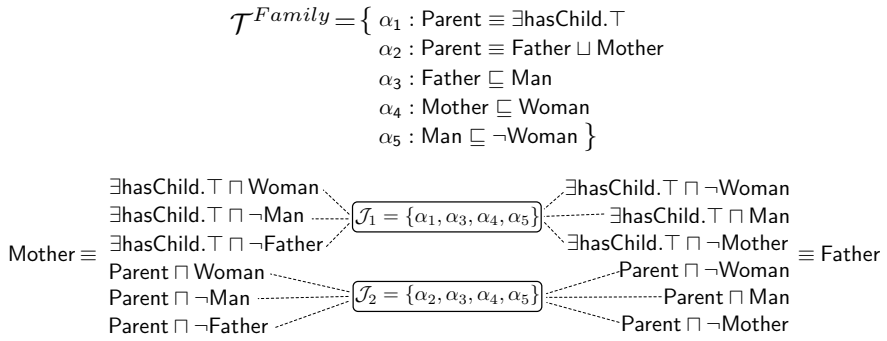

Figure 1: This small ontology describes a family domain. Concepts Mother and Father are implicitly defined in $\mathcal{T}^{Family}$, hence these are also explicitly definable, as shown by their definition axioms. Each axiom is explained by a justification $(\mathcal{J}_1, \mathcal{J}_2)$, denoted with dashed line.

2.1 Deciding Definability

A particular concept name C can either be defined *explicitly* or *implicitly* under an ontology, or be *undefined*. Explicit definability is a syntactic notion; deciding whether C is explicitly defined under an ontology is the trivial process of searching the TBox for a concept equivalence axiom whose left-hand side is C, and the potentially complex concept on the right-hand side does not include C (e.g. $\mathsf{C} \equiv \mathsf{D}$ where $\mathsf{C} \notin \mathsf{Sig}(\mathsf{D})$). The process is formalised by Algorithm 1.

Algorithm 1: IsConceptExplicitlyDefined($\mathsf{C}, \mathcal{T}, \Sigma$)

Input : C: concept name, $\mathcal{T}$: TBox, Σ: signature
Output: *Boolean*: True if C is explicitly defined by Σ in $\mathcal{T}$, False otherwise

```
if C ≡ D ∈ T, where Sig(D) ⊆ Σ then
    return True
end
return False
```

In contrast, implicit definability is a semantic notion whose detection requires reasoning. Algorithm 2 implements the *ten Cate et al.* method for determining implicit definability of a concept using a particular signature [25]. The process reduces the definability check to an entailment check [26]. The computational complexity of determining whether a concept is implicitly defined depends on the complexity of the entailment check, which is predicted on the expressivity of the given DL language. Thus it is potentially exponential in the size of the ontology, for expressive DL dialects.

Algorithm 2: IsConceptImplicitlyDefined($\mathsf{C}, \mathcal{T}, \Sigma$)

Input : C: concept name, $\mathcal{T}$: TBox, Σ: signature
Output: *Boolean*: True if C is implicit defined, False otherwise

1 $\mathcal{T}' \leftarrow \mathcal{T}$
2 $\mathcal{K} \leftarrow \mathsf{Sig}(\mathcal{T}) \setminus \Sigma$
3 **for** $e \in \mathcal{T}\prime$ **do**
4 | **if** $e \in \mathcal{K}$ **then**
5 | | $e \leftarrow e'$
6 | **end**
7 **end**
8 **if** $\mathcal{T} \cup \mathcal{T}' \models \mathsf{C} \equiv \mathsf{C}'$ **then**
9 | **return** True
10 **else**
11 | **return** False
12 **end**

Algorithm 2 Walkthrough. First $\mathcal{T}'$, an identical copy of TBox $\mathcal{T}$ is created (line 1). Next, the entity set $\mathcal{K}$ is initialised, by taking the signature of the TBox, and reducing it with Σ, the set of entities allowed to be used in the complex concept which would define C (line 2). Now $\mathcal{K}$ only contains those entities of the TBox that would not be a part of a prospective non-primitive concept definition of C. At this point the process begins iterating through every occurrence of every entity in $\mathcal{T}'$ (i.e. in all the axioms of the ontology) to rename those entities that also appear in $\mathcal{K}$, thus each entity $e \in \mathcal{K}$ becomes e' in $\mathcal{T}'$.

After each entity has been renamed $\mathcal{T} \cap \mathcal{T}' = \Sigma$, i.e. the two TBox-es only share those entities that are allowed to be used in a definition of C. Next the axiom $\mathsf{C} \equiv \mathsf{C}'$ is created, which is an equality between the two versions of the concept in question (the original from $\mathcal{T}$, and the renamed in $\mathcal{T}'$). Finally the merged TBox $\mathcal{T} \cup \mathcal{T}'$ (that consists of the original, and the renamed version) is queried, whether this axiom is entailed by it (line 8). If it follows that means that C is indeed implicitly defined under $\mathcal{T}$, using only entities of Σ; otherwise it is not implicitly defined.

Optimisation. In order to reduce complexity the definability check process is optimised by first checking explicit definability, due to the fact that this approach has significantly lower computational complexity compared to the implicit definability check. In case the concept in question is not explicit defined, then it is checked for implicit definability. If both definability checks fail, the concept is clearly undefined in the ontology.

2.2 Justifying Definability

It is often difficult for humans to identify the axiom set in a TBox that implies definability. *Justifications* [12] can be used to validate definability and to provide a set of axioms supporting an entailment. A justification $\mathcal{J}$ for an entailment

η in an ontology is the ontological fragment in which η holds (i.e. a set of TBox axioms such that $\mathcal{J} \subseteq \mathcal{O}$). A justification is *minimal*, if the entailment in question does not follow from any proper subset of the justification. For example, if we assume $\mathcal{O} = \{\mathsf{A} \sqsubseteq \mathsf{B}, \mathsf{B} \sqsubseteq \mathsf{C}, \mathsf{D} \sqsubseteq \exists \mathsf{r}.\mathsf{C}\}$ and the axiom $\alpha : \mathsf{A} \sqsubseteq \mathsf{C}$, then $\mathcal{O} \models \alpha$ holds as $\{\mathsf{A} \sqsubseteq \mathsf{B}, \mathsf{B} \sqsubseteq \mathsf{C}\} \subseteq \mathcal{O}$; i.e. the entailment is justified[4].

Algorithm 3: JustifyDefinability($\mathsf{C}, \mathcal{T}, \Sigma$)

Input : C: concept name, $\mathcal{T}$: TBox, Σ: signature
Output:

$\mathcal{T}' \leftarrow \mathcal{T}$
$\mathcal{K} \leftarrow \mathsf{Sig}(\mathcal{T}) \setminus \Sigma$
for $e \in \mathcal{T}'$ **do**
  if $e \in \mathcal{K}$ **then**
    $e \leftarrow e'$
  end
end
$\alpha \leftarrow \mathsf{C} \equiv \mathsf{C}'$
$\mathcal{J} \leftarrow$ComputeSingleJustification$((\mathcal{T} \cup \mathcal{T}'), \alpha)$
return $\mathcal{J}$

The algorithm checking for implicit definability (Algorithm 2) can be modified to compute not only whether a concept is definable with respect to a given signature, but to also provide its justifications; thus instead of directly querying a reasoner, the entailment check is delegated to the justification algorithm, that returns either a justification if the given signature defines the concept in question, or an empty set if the definability does not hold. This method formalised by Algorithm 3.

2.3 Definability of Roles

The notion of definability also applies to roles, hence a role can be classified either as *defined* (explicitly or implicitly), or as *undefined*. A defined role means that its extensions (set of pairs of individuals) can be unambiguously determined under an ontology, if the individuals of those entities that are used to define the role are known in an interpretation.

Equivalently to concepts, explicit role definability is a syntactic notion, expressed by a *single axiom*; whereas implicit definability is a semantic notion where the meaning is implied by a *set of axioms*. In the axiom

$$\mathsf{brotherOf} \equiv \mathsf{maleSiblingOf} \tag{1}$$

both brotherOf and maleSiblingOf are **explicitly defined** as *synonym role* of the other, using a role equivalence axiom. Similarly, the following two role inclusion

[4]Horridge et. al. [12] introduced an efficient approach that computes either a single, or all justifications of an entailment.

axioms define both roles as synonym, however in this case these are **implicitly defined**:

$$\{\mathsf{brotherOf} \sqsubseteq \mathsf{maleSiblingOf}, \mathsf{maleSiblingOf} \sqsubseteq \mathsf{brotherOf}\} \models \mathsf{brotherOf} \equiv \mathsf{maleSiblingOf} \tag{2}$$

In the next statement $\mathsf{parentOf}$ is explicitly defined as the inverse of the $\mathsf{childOf}$ role.

$$\mathsf{parentOf} \equiv \mathsf{childOf}^{-} \tag{3}$$

The following axiom set also explicitly defines the $\mathsf{parentOf}$ role:

$$\{\mathsf{parentOf} \equiv \mathsf{fatherOf} \sqcup \mathsf{motherOf}, \quad \mathsf{fatherOf} \sqcap \mathsf{motherOf} \sqsubseteq \bot\} \tag{4}$$

as well as implicitly define both $\mathsf{fatherOf}$ and $\mathsf{motherOf}$ as

$$\mathsf{motherOf} \equiv \mathsf{parentOf} \sqcap \neg\mathsf{fatherOf} \qquad \mathsf{fatherOf} \equiv \mathsf{parentOf} \sqcap \neg\mathsf{motherOf} \tag{5}$$

Although these example are all valid DL statements this form of axiomatisation is quite rare, for instance, in OWL2, which is one of the most used ontology representation language, the axiom set (5) cannot be expressed.

Deciding definability. Deciding role definability can be achieved by using the same method as for concepts (Algorithm 2), with the following modifications:

- Instead of operating on the entire TBox, the process must be restricted to the RBox (set of role axioms). TBox axioms, excluding the RBox, describe concepts (often by using roles). If the TBox is used to decide role definability, false positives may emerge. For example, the axiom $\exists \mathsf{s}.\top \equiv \exists \mathsf{r}.\top$, which implies that these roles r, s share the same domain concept, would be incorrectly identified as a definition axiom for both roles.
- The candidate definition signature should only contain role names, as roles are only defined in terms of other roles, i.e. $\Sigma \subseteq \mathsf{N_R}$.

In order to decide definability of a concept whose description contains defined roles, there is no need to separately assess definability of the role. For example, in the ontology fragment (6) concept $\mathsf{Document}$ is implicitly defined, either as the domain of role $\mathsf{used\_by}$, or the range of the inverse relation use.

$$\begin{aligned} \mathcal{T} = \{ & \mathsf{used\_by} \equiv \mathsf{use}^{-}, \\ & \mathsf{Document} \sqsubseteq \exists \mathsf{used\_by}.\mathsf{Person} \\ & \top \sqsubseteq \forall \mathsf{use}.\mathsf{Document}\} \end{aligned} \tag{6}$$

Prevalence. In practice, the most prevalent forms of defined roles are *synonym* an *inverse* roles. Role constructors, such as role hierarchy ($\mathcal{H}$), transitive role ($\mathcal{R}$) and complex role inclusion axioms ($\mathcal{R}$) are only part of (very) expressive DL languages, where the complexity of reasoning services considerably increases due to the use of constructors, making these languages less practical to use, and subsequently less prevalent in real-world scenarios, compared to weaker DLs.

3 Minimal Definition Signatures (MDSs)

This section defines the different concept definition signature types, explains why there could be an exponential number of MDSs for a given defined concept, how definition signatures can be validated, and shows that minimal definition signatures can be used to detect certain type of ontology modelling errors. As discussed in Section 2, every defined concept has a corresponding description(s). Each description (except explicitly defined synonyms) is a complex concept, i.e. an entity set conjoined by appropriate concept constructors, that assign precise meaning to the defined concept concept name, under an ontology.

$$\underbrace{\mathsf{Parent}}_{\text{defined concept}} \equiv \underbrace{\mathsf{Mother} \sqcup \mathsf{Father}}_{\text{concept description}} \tag{7}$$

The defined concept is the left-hand side (LHS), its description is the right-hand side (RHS) of a non-primitive concept definition axiom. A definition signature can be defined as:

Definition 3 ***Definition Signature (DS)*** *A set of entities names Σ is a definition signature of the concept C under a TBox $\mathcal{T}$, if and only if members of Σ can be used to construct the right-hand side of a definition axiom for C, i.e. there is some complex concept D, such that $\mathsf{Sig}(\mathsf{D}) \subseteq \Sigma$, and $\mathcal{T} \models \mathsf{C} \equiv \mathsf{D}$, where $\Sigma \subseteq \mathsf{Sig}(\mathcal{T}) \setminus \{\mathsf{C}\}$.*

If a concept C is defined in an ontology, then we can entail that there exists some subset of the ontology signature that implicitly defines C. We only focus on acyclic definitions, as definitions with *direct cycles* (where the defined concept appears in its corresponding description) are excluded by this definition.

As definition signatures may contain *redundant members*, their size could be very large, thus we introduced the notion of *signature minimality*:

Definition 4 ***Minimal Definition Signature (MDS)*** *A signature Σ is a minimal definition signature of a defined concept C under a TBox $\mathcal{T}$, if none of its proper subsets are definition signatures of C.*

The *minimality* property of an MDS refers to minimising the size of the signatures, by eliminating superfluous entities. However, a defined concept may have multiple *unique* MDSs (where the difference of any two MDSs is not an empty set) under an ontology, with the same cardinality. From the definition, it follows that every MDS is also a DS, and any DS may contain at least one, but potentially multiple MDSs. For example, in the $\mathcal{T}^{Family}$ example (Figure 1), the signature $\Sigma = \{\mathsf{hasChild}, \mathsf{Man}, \mathsf{Woman}\}$ is a DS of all three defined concepts in the TBox. However, this signature is not a minimal DS of Parent, because it can be defined by the following two MDSs: $\{\mathsf{hasChild}\}, \{\mathsf{Mother}, \mathsf{Father}\}$; as formalised by axiom α_1 and α_2, respectively.

Validity. An signature is a valid MDS if and only if it is:

- *correct:* it explicitly or implicitly defines a concept under a given TBox;

- *minimal:* it contains no redundant members;
- *acyclic:* it does not contain the concept name which it defines;
- *not empty:* the only concept which can be defined with an empty definition signature is $\top$.

Number of MDSs. The number of possible rewritings of a defined concept is exponential in the size of the ontology. This value is used as the *quantitative measure of the extant of concept definability.*

Descriptions of defined concepts are built inductively using other, potentially also defined concepts; therefore the number of possible concept rewritings are dependent on the definability of its constituent concepts. The definability of any defined description member concepts is dependent on the definability of its own description, i.e. *definability is a recursive notion.*

For example, let us consider a TBox (8), where A is a defined concept, such that each constituent concept B_i in the description of A is also defined.

$$\begin{aligned} \mathsf{A} &\equiv \mathsf{B}_1 \sqcup \ldots \sqcup \mathsf{B}_n \\ \mathsf{B}_i &\equiv \mathsf{C}_1 \sqcup \ldots \sqcup \mathsf{C}_m \qquad \text{where } (1 \leq i \leq n) \end{aligned} \tag{8}$$

Then A is defined by the signature $\Sigma^{\mathsf{A}} = \{\mathsf{B}_1, \ldots, \mathsf{B}_n\}$, and any other signature, that is the result of substituting any defined concept B_i in Σ^{A} with its corresponding definition ($\Sigma^{\mathsf{B}_i} = \{\mathsf{C}_1 \sqcup \ldots \sqcup \mathsf{C}_m\}$), hence the number of potential syntactically different definitions, as well as the number of corresponding MDSs is exponential.

The set of all MDSs of a particular concept may *overlap* (share one or more entities), or in other cases some MDSs may be *pairwise disjoint* (especially when the MDS contains only a single entity, such as when the MDS corresponds to a domain or range concept pattern, or an explicit synonym pattern).

3.1 MDS to identify modelling errors

MDSs can shed light on some modelling errors in an ontology. So far three type of errors were formalised, these are all automatically detectable, but repairing requires involvement of an ontology engineer and a domain expert.

3.1.1 Redundant concept(s).

An explicit concept definition should be a succinct representation, meaning that it should only consist of entities that are necessary to unambiguously describe the concept. If the signature of an explicit concept definition is not *minimal*; i.e. a subset of this signature (an MCDS) can be used to define the concept, it implies a discrepancy between the *intended meaning* (formalised by the explicit definition axiom), and the *actual meaning* (an alternative explicit definition axiom corresponding to an MCDS of the defined concept). Any redundant concept in the definition is semantically ignored.

Example 1 [5] *In this example* Regular_author *is explicitly defined by* α_1.

$$\begin{aligned}\mathcal{J} = \{\alpha_1 : \mathsf{Regular\_author} \equiv (\mathsf{Contribution\_1th} - \mathsf{author} \sqcup \mathsf{Contribution\_co} - \mathsf{author}) \\ \sqcap (\underbrace{\exists \mathsf{contributes.Conference\_contribution}}_{redundant}) \\ \alpha_2 : \mathsf{Contribution\_1th} - \mathsf{author} \sqsubseteq \mathsf{Regular\_author}, \\ \alpha_3 : \mathsf{Contribution\_co} - \mathsf{author} \sqsubseteq \mathsf{Regular\_author}\}\end{aligned} \tag{9}$$

However, this explicit definition signature is not minimal (i.e. is a CDS but not an MCDS), as its subset $\{\mathsf{Contribution\_1th} - \mathsf{author}, \mathsf{Contribution\_co} - \mathsf{author}\}$ *can be used to define* Regular_author *by the axiom*

$$\mathsf{Regular\_author} \equiv \mathsf{Contribution\_1th} - \mathsf{author} \sqcup \mathsf{Contribution\_co} - \mathsf{author}$$

as it is implied by the justification axiom set $\mathcal{J}$. *The error arise because:*

- $\alpha_1 \models \mathsf{Regular\_author} \sqsubseteq (\mathsf{Contribution\_1th} - \mathsf{author} \sqcup \mathsf{Contribution\_co} - \mathsf{author})$,
- $\{\alpha_2, \alpha_3\} \models (\mathsf{Contribution\_1th} - \mathsf{author} \sqcup \mathsf{Contribution\_co} - \mathsf{author}) \sqsubseteq \mathsf{Regular\_author}$

Clearly the anomy concept $\exists$contributes.Conference_contribution *is redundant in* α_1. *This is most likely not what the ontology engineer intended.*

3.1.2 Unwanted synonym(s).

This can occur when two or more concepts that suppose to convey different meaning are wrongly represented as interchangeable synonyms of one another.

Example 2 [6] *This example shows three different ways of defining the concept* Anthropometrics_Height.

- $\mathcal{J}_1 \models \mathsf{Anthropometrics\_Height} \equiv \mathsf{Anthropometrics\_Weight}$
- $\mathcal{J}_2 \models \mathsf{Anthropometrics\_Height} \equiv \mathsf{Anthropometrics\_BMI}$
- $\mathcal{J}_3 \models \mathsf{Anthropometrics\_Height} \equiv \mathsf{Anthropometrics}$

Obviously, Anthropometrics_Height, Anthropometrics_Weight *and* Anthropometrics_BMI *are semantically related, but otherwise completely different concepts. However, in TBox* $\mathcal{T}$ *where* $(\mathcal{J}_1 \cup \mathcal{J}_2 \cup \mathcal{J}_3) \subseteq \mathcal{T}$, *these concepts are defined as equivalent.*

$$\begin{aligned}\mathcal{J}_1 = \{\alpha_1 : \mathsf{Anthropometrics} \sqsubseteq \mathsf{Anthropometrics\_BMI} \sqcap \\ \mathsf{Anthropometrics\_Height} \sqcap \mathsf{Anthropometrics\_Weight}, \\ \alpha_2 : \mathsf{Anthropometrics\_Height} \sqsubseteq \mathsf{Anthropometrics}, \\ \alpha_3 : \mathsf{Anthropometrics\_Weight} \sqsubseteq \mathsf{Anthropometrics}\}\end{aligned} \tag{10}$$

[5] CONFERENCE corpus, conference.owl
[6] BIOPORTAL corpus, *bp26.owl*

$$\mathcal{J}_2 = \{\alpha_1 : \mathsf{Anthropometrics} \sqsubseteq \mathsf{Anthropometrics\_BMI} \sqcap \mathsf{Anthropometrics\_Height} \sqcap \mathsf{Anthropometrics\_Weight},$$
$$\alpha_2 : \mathsf{Anthropometrics\_Height} \sqsubseteq \mathsf{Anthropometrics}\}$$

$$\mathcal{J}_3 = \{\alpha_1 : \mathsf{Anthropometrics} \sqsubseteq \mathsf{Anthropometrics\_BMI} \sqcap \mathsf{Anthropometrics\_Height} \sqcap \mathsf{Anthropometrics\_Weight},$$
$$\alpha_2 : \mathsf{Anthropometrics\_Height} \sqsubseteq \mathsf{Anthropometrics},$$
$$\alpha_4 : \mathsf{Anthropometrics\_BMI} \sqsubseteq \mathsf{Anthropometrics}\}$$

The correction requires some expert knowledge. Anthropometrics *means measurement of the size and proportions of the human body* [7]*. Axioms* $\alpha_2, \alpha_3, \alpha_4$ *are correct, as height, weight and BMI are all type of measurements that make up the general class* Anthropometrics*, but axiom* α_1 *is incorrect as height and weight measurements would share nothing in common (their intersection would be empty). The correct representation would be to describe* Anthropometrics *as a disjoint union of these concepts.*

3.1.3 Implicitly defined by an empty MCDS.

The only concept in any ontology, which requires no signature for its definition is $\top$. If a named concept is definable by an empty signature, then the ontology is most likely to contain an error, or purposely define the concept as the synonym of $\top$. For example in the *cocus.owl* ontology of the CONFERENCE corpus, concept Person is equivalent to $\top$. By examining the document, it becomes obvious that this is unintentional, as the ontology contains many other concepts (such as Conference) that are definitely not semantically related to Person.

4 Computing Minimal Definition Signatures

Finding MDSs can be computationally expensive, as the number of definitions themselves can be exponential in the size of the ontology. Furthermore, the set of candidate signatures is equivalent to the power set of the TBox signature (excluding the defined concept itself, i.e. $2^{\mathsf{Sig}(\mathcal{T})\setminus\{\mathsf{C}\}}$). Moreover, each candidate signature must be subjected to an *implicit definability check*, where the complexity of checking each case of definability is predicated on the DL expressivity of the ontology language, thus it can be exponential in the size of the ontology, for more expressive DL flavours.

However, typically the average case complexity is expected to stay within manageable bounds, thus it is feasible to compute MDS because: (1) In general the *number of MDSs are relatively low.* Defined concepts (and roles) are described in terms of other entities, hence the number of MDSs of a given concept depends on the number of MDSs of its description entities. Therefore the number of "less-defined" entities (entities with relatively low number of MDSs) is considerably larger than the "more-defined" ones. (2) In order to reduce

[7] `http://dictionary.reference.com/browse/anthropometric`

this complexity, *modularisation* [19] is used as space reduction mechanism. In the most general sense, a module $\mathcal{M}$ is an ontology fragment ($\mathcal{M} \subseteq \mathcal{T}$) which functions independently from the ontology, with respect to some subject matter, e.g. a signature, or an axiom set. In other words, a module preserves all entailments over a signature $\mathcal{S}$, which, in this context, is always the singleton signature consisting of a concept name, i.e. $\mathcal{S} = \{\mathsf{A}\}$. As modules preserve all entailments with respect to a signature, any MDS of a defined concept is contained in the module signature. *Syntactic locality based modules* (LBM) have been shown to be sound approximations of *semantic locality based modules* (that preserve entailments over all the terms that occur in the module) [5], and there are efficient and widely used polynomial time algorithms for extracting syntactic LBMs[8]. Furthermore, there are efficient and widely used polynomial time algorithms for extracting syntactic LBMs. As modules can be considerably smaller compared to the original ontology, modularisation is an effective mechanism for reducing the complexity of computing MDSs. The initial search space $\mathcal{P}(\mathsf{Sig}(\mathcal{T}) \setminus \{C\}))$ can be reduced by using *modularisation*. (3) Implicit definability check is performed by a *reasoner*. In practice, state of the art systems has been shown to work for most of the currently existing TBoxes, even for more expressive DL languages.

In the following, various MDS computation algorithms are presented. Each algorithm is used to achieve a particular task (such as computing a single MDS, disjoint MDSs etc.), where the composition of these methods form the approach that computes all MDSs of a defined concept. Computing all MDSs (Section 4.4) is a two phase process, first a set of pairwise disjoint MDSs need to be acquired (Section 4.2), which can be achieved in logarithmic to polynomial time (depending on the particular defined entity, the number of its corresponding disjoint MDSs, and the method used to compute the MDSs), then the set of disjoint MDSs is expanded (Section 4.3) until the complete set of different MDSs are identified, the complexity of the later is potentially exponential in the size of the ontology (more precisely, the size of the module).

4.1 Computing a Single MDS

There are two approaches to compute a single MDS, the *Single entity pruning* (Section 4.1.1) algorithm always runs in linear time, whilst the *divide and conquer* (D&Q) (Section 4.1.2) algorithm at best takes logarithmic time to compute. The later approached is favoured when the input signature is large (i.e. it consists of the entire signature of the ontology, or the module which describes a defined entity). The former algorithm is used in optimisation, when computing MDSs of an explicitly defined concept, the first candidate signature is the explicit definition(s), such signatures are often already minimal definition signatures, or definition signatures that contain only a small number of redundant entities. In this case the D&Q approach performs at its worst, taking twice as long as the single entity pruning approach.

[8]The OWL API provides methods for extracting several types of LBMs.

4.1.1 Single Entity Pruning

This algorithm finds *one MDS* of a given defined concept, which is contained within the input signature, assuming that this signature implicitly defines the concept under the ontology. Figure 2 depicts the process formalised by Algorithm 4.

Algorithm 4: ComputeSingleMDS($\mathsf{C}, \mathcal{T}, \mathcal{S}$)

Input : C: defined concept; $\mathcal{T}$: TBox; $\mathcal{S}$: signature (set of entities)
Output: Σ: one minimal definition signature of concept C

1 $\Sigma \leftarrow \mathcal{S}$
2 **for** $\mathsf{e} \in \Sigma$ **do**
3 $\quad \Sigma \leftarrow \Sigma \setminus \{\mathsf{e}\}$
4 $\quad$ **if** IsConceptImplicitlyDefined($\mathsf{C}, \mathcal{T}, \Sigma$) *is False* **then**
5 $\quad\quad \Sigma \leftarrow \Sigma \cup \{\mathsf{e}\}$
6 $\quad$ **end**
7 **end**
8 **return** Σ

Walkthrough. The basic idea behind the algorithm is to systematically prune the input signature $\mathcal{S}$ until it contains no redundant members, thus it becomes *minimal*, whilst still implicitly defining concept C by the signature $\mathcal{S}$, under a TBox $\mathcal{T}$. The *prospective* minimal signature set Σ is initially set to be equal to the *input* signature $\mathcal{S}$ (line 1). Next, each entity $\mathsf{e} \in \Sigma$ is tested whether it is required to be the member of the minimal signature (line 2-7). First the entity is removed the from the signature (line 3), then Σ is tested whether the signature is still correct, i.e. if it implicitly defines C (line 4). If Σ is no longer a correct concept definition signature, it means that e was a required member of Σ, therefore it is put back in Σ (line 5), otherwise it is redundant and it remains discarded. The algorithm terminates when all members of the input signature $\mathsf{e} \in \Sigma$ have been examined. At this point Σ is a minimal concept definition signature of C, and it is returned by the algorithm (line 8).

If the input signature $\mathcal{S}$ contains more than one MDSs, then the outcome (resulting signature set) depends on the entity ordering in $\mathcal{S}$. In order words, the same input can yield different solutions if the ordering of entities in $\mathcal{S}$ is changed, but the same ordering always yields the same output.

Correctness. The only outcome of the process is that the input signature, which is a concept definition signature (DS) of C, becomes minimal (i.e. an MDS):

- the *precondition* is that C is implicitly defined by $\mathcal{S}$ under $\mathcal{T}$, thus it is already correct ($\mathcal{T} \models \mathsf{C} \equiv \mathsf{D}$, where $\mathsf{Sig}(\mathsf{D}) \subseteq \mathcal{S}$);
- the *postcondition* is that the output Σ is minimal, this is achieved by removing all those entities that without Σ can still be used to describe C

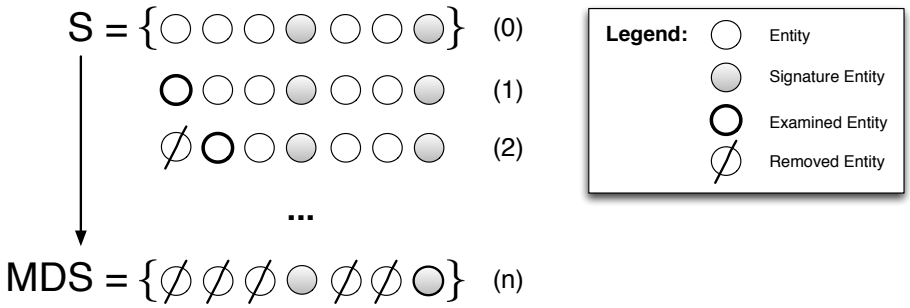

Figure 2: Computing an MDS using the single entity pruning method.
($\mathcal{T} \models \mathsf{C} \equiv \mathsf{D}$, where $\mathsf{Sig}(\mathsf{D}) \subseteq \Sigma$).

The difference between the input $\mathcal{S}$, and the output Σ, is a set of redundant definition signature entities (discarded members from the input $\mathcal{R} = \mathcal{S} \setminus \Sigma$).

Termination and Completeness. Each member of the input $\mathsf{e} \in \Sigma$ is examined once, therefore it terminates when all elements of this set are exhausted. Completeness, i.e. finding one minimal CDS is ensured by the pre, and post-condition.

Complexity. The process takes n steps to complete, where n is the number of entities found in the input signature ($n = |\mathcal{S}|$). Thus the asymptotic complexity is linear ($O(n)$), in the size of the input signature, not including the complexity of the implicit definability check of each candidate signature. Regardless of the number of required signature entities, the complexity is always $O(n)$, because each entity is examined exactly once.

4.1.2 Divide and Conquer

Although the previously presented approach (Algorithm 4) has polynomial worst-case time complexity, for large signatures, the process can still take considerable time, because every entity is individually examined by performing an implicit definability check, which is a time consuming operation (potentially exponential). In order to improve performance, the number of implicit definability checks need to be reduced. This is achieved by examining entities in groups, using a *top-down style "divide and conquer"* strategy. The method is depicted by Figure 3 and formalised in Algorithm 5 and 6.

Divide and Conquer. The divide and conquer algorithm design paradigm is commonly used in computer science [4]. The approach solves difficult problems by recursively splitting them into sub-problems until each part becomes simple enough to be solved. The final solution is derived by combining the results of sub-problems. Some applications of this approach include sorting (e.g. merge sort), searching (e.g. binary search), syntactic analysis (e.g. top-down parsers).

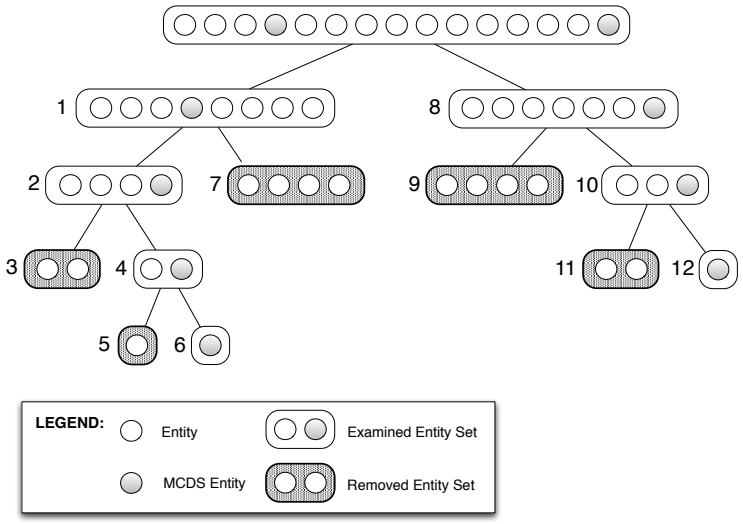

Figure 3: Computing an MDS using the divide and conquer method. Tree nodes are labelled according to the order of traversal.

The strategy is applied to MDSsearch in the following way: instead of examining candidate signature entities individually, entities are tested as groups to see whether they are required members of the MDS. Each entity group is recursively split until a smaller subset is found, such that it is either *removable*, or cannot be split any further (i.e. contains only one entity which is a *required* MDS member). The process generates a *binary-tree*, where the root node is the input signature, and every other node is a subset of the input signature. Every leaf node is either a set of removable entities, or a single entity required in the final signature.

First a given signature $\mathcal{S}$ (which is a definition signature, but not yet necessarily minimal) is split into two halves, $\mathcal{S}_L$ and $\mathcal{S}_R$. Next $\mathcal{S}_L$ is checked: if the signature $\mathcal{S} \setminus (\mathcal{R} \cup \mathcal{S}_L)$ can still implicitly define the given concept, then $\mathcal{S}_L$ contained no required signature member(s) and it is permanently added to $\mathcal{R}$, the set of removable entities of $\mathcal{S}$. The same test is also done for the other half: $\mathcal{S}_R$. Otherwise, if the implicit definability check fails, it means that $\mathcal{S}_L$ contains required signature member(s), thus it cannot be removed as a whole. At this point a recursive call is made, $\mathcal{S}_L$ is split into halves and the above process is repeated.

In oppose to single entity pruning, with this approach the input signature is not pruned during the process, due to the use of recursion. Instead the removable entities are recoded in $\mathcal{R}$.

Walkthrough. The ComputeSingleMDS-D&C (Algorithm 5) serves as a "runner" for its recursive subroutine SplitAndPrune (Algorithm 6). After initialisation (line 1-2), ComputeSingleMDS-D&C calls SplitAndPrune to identify $\mathcal{R}$, the set of redundant members of $\mathcal{S}$ (where $\mathcal{R} \subseteq \mathcal{S}$). When $\mathcal{R}$

Algorithm 5: ComputeSingleMDS-D&C($\mathsf{C}, \mathcal{T}, \mathcal{S}$)

Input : C: defined concept; $\mathcal{T}$: TBox; $\mathcal{S}$: signature
Output: Σ: one minimal definition signature of concept C

1 $\mathcal{S}' \leftarrow \mathcal{S}$
2 $\mathcal{R} \leftarrow \emptyset$
3 $\mathcal{R} \leftarrow$ SplitAndPrune($\mathsf{C}, \mathcal{T}, \mathcal{S}', \mathcal{S}, \mathcal{R}$)
4 $\Sigma \leftarrow \mathcal{S} \setminus \mathcal{R}$
5 **return** Σ

Algorithm 6: SplitAndPrune($\mathsf{C}, \mathcal{T}, \mathcal{S}', \mathcal{S}, \mathcal{R}$)

Input : C: defined concept; $\mathcal{T}$: TBox; $\mathcal{S}'$: examined signature part; $\mathcal{S}$: original signature; $\mathcal{R}$: removable entities

1 **if** $|\mathcal{S}'| > 1$ **then**
2 $\quad \mathcal{S}_L, \mathcal{S}_R \leftarrow$ Split($\mathcal{S}'$)
3 $\quad \mathcal{S}_{check} \leftarrow \mathcal{S} \setminus (\mathcal{R} \cup \mathcal{S}_L)$
4 $\quad$ **if** IsConceptImplicitlyDefined($\mathsf{C}, \mathcal{T}, \mathcal{S}_{check}$) *is True* **then**
5 $\quad\quad \mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{S}_L$
6 $\quad$ **end**
7 $\quad$ **else**
8 $\quad\quad \mathcal{R} \leftarrow$ SplitAndPrune($\mathsf{C}, \mathcal{T}, \mathcal{S}_L, \mathcal{S}, \mathcal{R}$)
9 $\quad$ **end**
10 $\quad \mathcal{S}_{check} \leftarrow \mathcal{S} \setminus (\mathcal{R} \cup \mathcal{S}_R)$
11 $\quad$ **if** IsConceptImplicitlyDefined($\mathsf{C}, \mathcal{T}, \mathcal{S}_{check}$) *is True* **then**
12 $\quad\quad \mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{S}_R$
13 $\quad$ **end**
14 $\quad$ **else**
15 $\quad\quad \mathcal{R} \leftarrow$ SplitAndPrune($\mathsf{C}, \mathcal{T}, \mathcal{S}_R, \mathcal{S}, \mathcal{R}$)
16 $\quad$ **end**
17 **end**
18 **return** $\mathcal{R}$

is computed, the initial signature $\mathcal{S}$ is pruned by removing $\mathcal{R}$ (line 4). The resulting signature Σ is now a minimal DS of C, which is then returned and the process terminates (line 5).

SplitAndPrune takes an input signature $\mathcal{S}'$ (where $\mathcal{S}' \subseteq \mathcal{S}$) and splits it into two halves $\mathcal{S}_L, \mathcal{S}_R$ (line 2). Next it generates the entity set $\mathcal{S}_{check}$ by taking the original signature ($\mathcal{S}$) and removing: (1) $\mathcal{R}$, i.e. all redundant entities found so far; (2) $\mathcal{S}_L/\mathcal{S}_R$ one half of the currently examined signature part. $\mathcal{S}_{check}$ is then tested to see whether it is a valid definition signature. In case it is valid, then every member of $\mathcal{S}_L/\mathcal{S}_R$ were *redundant entities*, thus these are added to $\mathcal{R}$. If $\mathcal{S}_{check}$ fails the definability check, then $\mathcal{S}_L/\mathcal{S}_R$ must contain at least one *required* entity (member of the final signature), hence it cannot be removed as a whole, thus SplitAndPrune is called to identify redundant members of $\mathcal{S}_L/\mathcal{S}_R$.

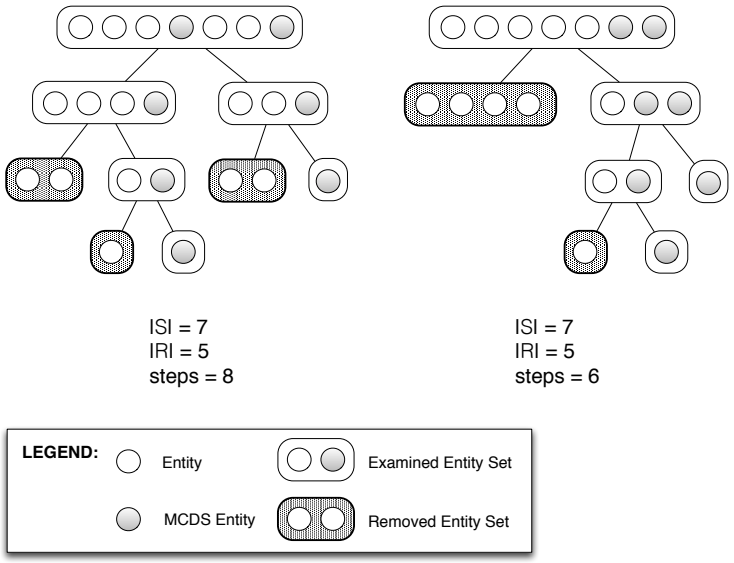

Figure 4: Computing MDSs with divide and conquer approach of two different signature orderings. $|\mathcal{S}|$ denotes the signature set cardinality, $|\mathcal{R}|$ denotes the number of redundant signature members. Both signatures are identical in size and redundant-required entity ratio, however due to the ordering, the process takes less time for the left-hand side signature.

Complexity. The complexity depends on the ratio of required and redundant members in the signature, and the ordering of the signature. Figure 4 shows how different signature orderings influence the process. The worst case occurs when all members $\mathsf{e} \in \mathcal{S}$ are required MDS entities. The required number of steps are $(2 * n) - 2$, where $n = |\mathcal{S}|$, i.e. it has linear time complexity. In the best case, when the signature contains only one required entity, the number of required steps (implicit definability checks) are $2 * (\lfloor log_2 n \rfloor)$, where $n = |\mathcal{S}|$ for all $|\mathcal{S}| \geq 4$. For example, given a signature where $|\mathcal{S}| = 1024$, it takes 20 implicit definability checks to identify the MDS member.

4.2 Computing Pairwise Disjoint MDSs

Algorithm 7 (depicted in Figure 5) builds upon single MDS computation methods in order to compute a *set of mutually disjoint MDSs* of a particular defined concept.

Walkthrough. The basic idea behind the algorithm is that at every iteration, a single σ (i.e. an MDS of C) is computed (line 3), then subsequently removed from the working signature $\mathcal{S}$ (line 4), until $\mathcal{S}$ contains no more MDSs, i.e. it does no longer implicitly defines C under $\mathcal{T}$, hence the process terminates.

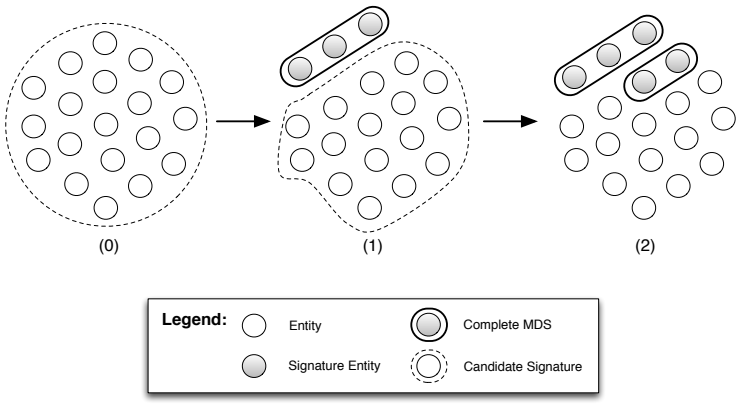

Figure 5: Computing mutually disjoint MDSs. Each iteration reduces the candidate signature.

Algorithm 7: ComputeDisjointMDSs($\mathsf{C}, \mathcal{T}, \mathcal{S}$)

Input : C: defined concept; $\mathcal{T}$: TBox; $\mathcal{S}$: signature (set of entities)
Output: $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$: a set of pairwise disjoint MDSs of C

```
Σ ← ∅
while IsConceptImplicitlyDefined(C, T, S) is True do
    σ ← ComputeSingleMDS-D&C(C, T, S)
    S ← S \ σ
    Σ ← Σ ∪ {σ}
end
return Σ
```

Termination. The algorithm operates on a working signature $\mathcal{S} = \{\mathsf{e}_1, \ldots, \mathsf{e}_n\}$, which contains a finite number of entities. This signature contains a finite number of MDS (set of entities). At each step, $\mathcal{S}$ is reduced by removing the constituent entities of the last found MDS. Once there are no more remaining MDSs, the working signature set $\mathcal{S}$ fails the implicit definability check, thus the process terminates.

Completness. The algorithm finds *a set of MDSs*, that are pairwise disjoint, but it does *not necessarily find* all disjoint MDSs. This depends on the following properties

- the *ordering of entities* in the set $\mathcal{S}$: the same input can yield different solutions, by changing the ordering of $\mathcal{S}$, however the same ordering always yields the same output.
- whether some of all the possible MDSs of a given concept overlap: if a concept only has *pairwise disjoint* MDSs, then the algorithm is *guaranteed to find them all*; otherwise any overlapping MDSs would not be found.

Complexity. The complexity the algorithm is *polynomial* in the size of the input signature ($O(n)$ where $n = |\mathcal{S}|$), this is determined by the following factors: (a) the time taken by its sub-routine ComputeSingleMDS-D&C (Algorithm 4), which is *logarithmic* in the size of input signature; (b) the (finite) number of iterations of the main loop where the sub-routine is called, which is equivalent to the number of mutually disjoint MDSs contained in the input signature number.

4.2.1 Optimisation

Algorithm 8: ComputeDisjointMDSs2($\mathsf{C}, \mathcal{T}$)

Input : C: defined concept; $\mathcal{T}$: TBox
Output: $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$: disjoint minimal concept defintion signatures of C

1 $\Sigma \leftarrow \emptyset$
2 $\mathcal{S} \leftarrow (\mathsf{Sig}(\mathcal{T}) \setminus \mathsf{C})$
3 **if** IsConceptDefined($\mathsf{C}, \mathcal{T}$) *is* ***Explicit*** **then**
4 $\quad \mathcal{A} \leftarrow$ GetExplicitDefinitionAxioms($\mathsf{C}, \mathcal{T}$)
5 $\quad$ **for** $\alpha \in \mathcal{A}$ **do**
6 $\quad\quad \sigma \leftarrow$ ComputeSingleMDS-D&C($\mathsf{C}, \mathcal{T}, (\mathsf{Sig}(\alpha) \setminus \mathsf{C})$)
7 $\quad\quad \Sigma \leftarrow \Sigma \cup \{\sigma\}$
8 $\quad\quad \mathcal{S} \setminus \mathsf{Sig}(\alpha)$
9 $\quad$ **end**
10 **end**
11 $\Sigma \leftarrow \Sigma \cup$ ComputeDisjointMDSs($\mathsf{C}, \mathcal{T}, \mathcal{S}$)
12 **return** Σ

In case a concept (for which the MDSs are to be determined) is *explicitly defined*, the process can be optimised by first computing MDSs from explicit definition axioms. At the worst case, locating an axiom has polynomial time complexity in $\mathsf{Ax}(\mathcal{T})$.

The method is formalised in Algorithm 8. First the explicit definition axioms are identified (line 4), then from each axiom signature an MDS is computed (note that not every definition axiom signature is a minimal DS) (line 6). Each MDS is taken out from the working signature set $\mathcal{S}$. Finally, by using the reduced input signature, any remaining pairwise disjoint MDSs are computed (line 11). Note that if a concept has multiple explicit definitions, where these signatures overlap, the resulting MDSs computed by this approach are not always pairwise disjoint.

4.3 Expanding MDSs

After computing a set of pairwise disjoint MDSs, any unidentified MDS must overlap with existing MDSs, i.e. combine entities from existing MDSs, and those entities that are not part of any identified MDS (the remainder of the TBox

signature)[9]. Finding overlapping MDSs requires a different search strategy than single, or pairwise disjoint MDS search, this is formalised by Algorithm 9. The algorithm has two applications: (1) it determines whether a given set of MDSs is complete, i.e. are there any unidentified MDSs of the given defined concept in the ontology; (2) using existing MDSs, it computes *some* previously unknown MDSs, thus expanding the prior MDS set.

Algorithm 9: ExpandMDSs($\mathsf{C}, \mathcal{T}, \Sigma$)

Input : C: defined concept; $\mathcal{T}$: TBox; $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$: set of *already identified* MDSs of C

Output: a potentially updated Σ is returned that may contain new MDSs, if the inputted Σ was incomplete

1 $\mathcal{S} \leftarrow \bigcup_{i=1}^{|\Sigma|} \sigma_i \{\forall \sigma_i \in \Sigma | 1 < |\sigma_i|\}$
2 $\mathcal{K} \leftarrow \mathsf{Sig}(\mathcal{T}) \setminus (\mathcal{S} \cup \{\mathsf{C}\})$
3 $\mathcal{S}' \leftarrow \mathcal{P}(\mathcal{S}) \setminus \Sigma$
4 **for** $\mathsf{s} \in \mathcal{S}'$ **do**
5 | $\mathcal{W} \leftarrow \mathcal{K} \cup \mathsf{s}$
6 | **if** IsConceptImplicitlyDefined($\mathsf{C}, \mathcal{T}, \mathcal{W}$) **then**
7 | | $\sigma \leftarrow$ ComputeSingleMDS($\mathsf{C}, \mathcal{T}, \mathcal{W}$)
8 | | $\Sigma \leftarrow \Sigma \cup \{\sigma\}$
9 | **end**
10 **end**
11 **return** Σ

Walkthrough. The prerequisite is that $|\Sigma| \geq 0$, i.e. there are some previously computed MDSs of C. If $\Sigma = \emptyset$, then the process terminates without computing any MDSs. First, all existing MDSs are merged into the set $\mathcal{S}$ (line 1), excluding MDSs containing only one entity; by definition single entity MDSs cannot be part of any minimal DS. Next, the signature of the TBox is deducted by $\mathcal{S} \cup \{\mathsf{C}\}$. The resulting entity set $\mathcal{K}$ now only contains those entities of the ontology which do not appear in any of the identified MDSs, $\sigma \in \Sigma$ (line 2). The initialisation concludes by computing $\mathcal{S}'$, the power set of $\mathcal{S}$ and reducing it with Σ (line 3).

Next the process begins to exhaustively test each subset $\mathsf{s} \in \mathcal{S}'$. From every s, a candidate definition signature $\mathcal{W}$ is generated by merging s with $\mathcal{K}$ (line 5). $\mathcal{W}$ is then examined to establish whether it implicitly defines C. If it does then $\mathcal{W}$ contains at least one new MDS. In this case a new MDS σ is extracted and stored in Σ (line 7-8).

At the end of the process Σ is returned. This either contains new MDSs, thus the input was incomplete, or is the same size as at input (i.e.the initial Σ contained all MDSs of C). If Σ contains new MDSs, it indicates that there may

[9]Due to the nature of the process (combining MDS and non-MDS entities to gain new ones), *no new mutually disjoint MDS* will be identified.

be more undiscovered MDSs. In this case the algorithm should be run again using the updated set of MDSs (the returned Σ) to check whether the updated Σ is complete.

Termination and Complexity. As the process includes exhaustively iterating through a power set, it has exponential computational complexity in $\mathcal{S}$ (the union of all MDSs). It terminates when all cases have been examined.

4.4 Computing All MDSs (Search & Expand)

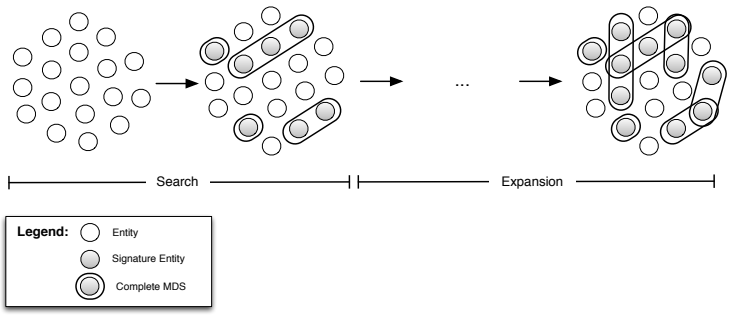

Figure 6: Depiction of a *Search and Expand* MDS computation strategy.

Algorithm 10 identifies *all* MDSs of a given concept by using a *search then expand* strategy, illustrated in Figure 6. During the *search* phase, it computes a set of mutually disjoint MDSs (using the ComputeDisjointMDSs algorithm). Then it proceeds with the *expansion* phase, which is an indefinite loop, where the process incrementally expands the set of MDSs by combining entities of existing MDSs and other ontology signature entities (using ExpandMDSs as its sub-routine). Although the expansion phase has exponential worsts case time complexity, evaluation suggests that this method is feasible to use for most real-world ontologies.

Walkthrough. The process begins by initialising entity set $\mathcal{S}$, which consists of the TBox signature, excluding the defined concept C (line 1). Then it computes a set of mutually disjoint MDSs of C, and stores it in Σ (line 2). Next, it enters a loop, which only terminates when all MDSs of C have been found (line 3-11). At each iteration, Σ is tested for completeness, using Algorithm 9 as a subroutine (line 4). If the returned Σ' is equivalent to Σ, then Σ was complete, thus the loop terminates and the complete set of MDSs are returned (line 6). Otherwise Σ' contains newly found MDSs, that are added to Σ (line 9), and the loop enters the next iteration repeating the above process.

Completeness and Termination. Completeness is ensured by (a) the ExpandMDSs algorithm (9), which is proven to be correct in identifying com-

Algorithm 10: ComputeAllMDSs-S&E($\mathsf{C}, \mathcal{T}$)

Input : C: defined concept; $\mathcal{T}$: TBox
Output: $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$: all MCDSs of C

```
1  S ← Sig(T) \ {C}
2  Σ ← ComputeDisjointMCDSs(C, T, S)
3  while True do
4  |  Σ' ← ExpandMDSs(C, T, Σ)
5  |  if Σ' = Σ then
6  |  |  return Σ
7  |  end
8  |  else
9  |  |  Σ ← Σ'
10 |  end
11 end
```

pleteness of an MDS set; (b) the loop structure, which only terminates when there are no more unidentified MDSs. Termination is ensured by the fact that the loop is not repeated unless new MDSs are found during the last iteration.

Complexity. Computational complexity of the algorithm is determined by two factors. Firstly, it depends on the number of all MDSs of the particular defined concept, because the process only terminates when the set of identified MDSs is complete. Although the number of MDSs is exponential in the size of the ontology, according to the experimental evaluation, this case rarely occurs in real-world ontologies.

$$\bigcup_{i=1}^{|\Sigma|} \sigma_i \in \Sigma \;\subseteq\; \mathsf{Sig}(\mathcal{T}) \setminus \{\mathsf{C}\} \text{ where } \{\forall \sigma_i \in \Sigma | 1 < |\sigma_i|\} \tag{11}$$

Secondly, as described in Section (4.3), the expansion phase involves exhaustively processing a power set of the union (11) of all priori computed MDS entities. However this is still significantly lower compared to using a naive brute-force approach, which would iterate trough all subsets of the power-set of the ontology (or module) signature. A brute force approach goes through the power set of the TBox signature ($\mathsf{Sig}(\mathcal{T} \setminus \{\mathsf{C}\})$), while this method only reaches that number, if the union of all MDSs of a given concept (11) is the same as the TBox signature. Evaluation suggests that this method is feasible to use because in most cases the union of all MDSs of a given concept (a) is not equivalent to the TBox signature; (b) its cardinality typically stays within reasonable bounds.

Replacing Defined Entities with MDSs. Algorithm 10 is clearly *unfeasible* to use, when the union of MDSs (11) is large. This scenario occurs if either there are many disjoint MDSs, the existing MDS are large, or the combination of both cases take place. Although the complete set of MDSs may not be computed in these cases, the following method provides a half measure. First the complete

set of MDSs should be computed for those defined entities that are feasibly computable. Then the disjoint MDSs of unfeasible entities can be expanded by replacing any defined entity in the disjoint MDSs with their corresponding MDSs.

5 Concept Definition Patterns (CDP-s)

In this section the different *axiomatisations of concept definitions are categorised into patterns*. An explicit concept definition is always formalised as a *single axiom*, whereas the definition of an implicitly defined concept is derived from an *axiom set* (i.e. a justification). Thus, in contrast with explicit definitions, implicit definitions are often not straightforward to recognise and understand. Therefore, a set of patterns were identified that facilitates the understanding of the relation between definition entities and their corresponding axioms.

Classification. Patterns are classified into basic and complex: *basic* patterns are atomic artefacts, whilst *complex* patterns are formed by the combination of basic ones. As previously described, Algorithm 3 makes use of justification-based explanation computation, that extracts a minimal axiom set (a justification $\mathcal{J}$) from the TBox which implies the meaning of a defined concept. By examining this axiom set ($\mathcal{J}$), and the concept definition signature (Σ), the corresponding concept definition pattern can be identified.[10]

Applications. CDPs can be used for the following tasks:

- Basic patterns serve as input for the *heuristics-based definition axiom generation* approach, which in contrast with general rewriting methods, is language independent and does not require ontological reasoning, therefore the computational complexity is linear in the size of the input. Although this method only covers basic patterns, the evaluation suggests that this is sufficient in many scenarios because the majority of all definability cases fall in to the basic pattern category.
- Axiom generation from complex patterns is not yet formalised, however the set of basic patterns (along with the corresponding axioms) forming a complex pattern can be pinpointed with this method, hence it can aid human comprehension thus simplify *manual axiom generation from complex patterns*.
- *Validation of concept definition signatures.*

[10]Please note that these patterns are *non-exhaustive*, i.e. the identified patterns are not guaranteed to represent all cases, however they cover all cases that emerged during the evaluation (thus as future work, a more fine-grade classification may be developed).

5.1 Related Work

As previously described, concepts are defined inductively using other concepts, roles and individuals from the ontology vocabulary. The number of different ways a definitions can be constructed is infinite. The actual content of a definition obviously depends on the defined concept, however the formalisation must follow the rules of the given ontology language. Every language uses the same three type of modelling primitives (i.e. entities), and has a restricted, finite set of logical operators available, thus it is possible to *generalise the common forms of creating concept definitions* (i.e. abbreviating complex concepts). The hereby presented definition patterns build on the following sources:

OWL API. The OWL API makes this generalisation explicit. In order to create an axiom in OWL API the user must choose the from a set of predefined axiom types (subclasses of the *AxiomType* Java class) [13]. For example, concepts that are mutually disjoint are formalised using the *OWLDisjointUnionAxiom* class, which translated to DL axioms as a concept equivalence axiom ($\mathsf{C} \equiv \mathsf{D_1} \sqcup \mathsf{D_2} \sqcup \mathsf{D_3}$) and a set of the corresponding axioms that state the disjoint relation between constituent entities of the concept description ($\{\mathsf{D_1} \sqsubseteq \neg \mathsf{D_2}, \mathsf{D_2} \sqsubseteq \neg \mathsf{D_3}\}$). This axiom type was adapted in the *Constituent concept of a disjoint union* CDP.

Ontology alignment design patterns. The various formalisations of definition axioms were studied in the area of *ontology alignment*. Matching concept and role names (i.e. generating simple correspondences) is a non-trivial, but well understood task, however matching a concept name to a complex concept is typically more challenging. Many state of the art approaches [20, 18, 24, 27] tackle the complex matching problem by making use of *patterns* in conjunction with other techniques (e.g. linguistic analysis [18], ontology transformation [24], etc.). Author of these publications argue that ontology engineering is a design problem, where ontology engineers often employ design patterns for formalising the content of a given knowledge base. In ontology alignment, patterns are meant to serve as generic solutions to particular ontology alignment problems, meaning that a mismatched (unmatched, or wrongly matched) entity may be fitted with a pattern and subsequently aligned to an entity of the other ontology (that is being aligned).

In [20] Scharffe et al. introduced the *Ontology alignment design patterns* library as a solution for recurring mismatches that can arise during the alignment of two ontologies. This repository (at the time of writing) consists of about a dozen alignment patterns, it is hosted online [11] as part of the collaboratively built (Wikipedia style) semantic web portal dedicated to ontology design patterns.

Let us consider the *Class by attribute value (CAV)* pattern, which occurs "when a class in one ontology corresponds to a class in the other ontology, given that the scope of the latter class is restricted to only those instances having a

[11] `http://ontologydesignpatterns.org/wiki/Submissions:AlignmentODPs`

specific value for a given attribute" [20]. For example, the following axiom

$$\mathsf{Late\_Registered\_Participant}_{\mathcal{O}_1} \equiv \exists \mathsf{earlyRegistration}_{\mathcal{O}_2}.\{\mathsf{false}\}$$

is an instantiation of the CAV pattern, where the concept Late_Registered_Participant of ontology $\mathcal{O}_1$, corresponds to the anonym (complex) concept $\exists$earlyRegistration.{false} of $\mathcal{O}_2$. This alignment pattern confirms that concepts are often described as the domain or range concept of some restriction; this was used in defining the *domain or range restriction concept of a role* CDP.

5.2 Pattern Recognition

Pattern recognition requires the following objects:

- C: the defined *concept name*;
- $\Sigma = \{\mathsf{e_1}, \ldots, \mathsf{e_n}\}$: the *concept definition signature*, consisting of a set of different entities, that can be used to create a complex concept, i.e. the right-hand side of an explicit definition axiom that describes the concept;
- $\mathcal{J} = \{\alpha_1, \ldots, \alpha_m\}$: the *justification*, consisting of a set of axioms that given a particular Σ, entails the definability of the concept.

The recognition is performed by analysing the *cardinality*, and the *type* of the constituent entities or axioms of the above sets, respectively.

5.3 Basic patterns

Basic patterns are trivial to recognise, as well as to use for constructing an explicit concept definition.

(1) Explicit definition patterns. These are patterns of explicitly defined concepts, that are always represented in form of a single, asserted concept equivalence axiom, hence $|\mathcal{J}| = 1$. However the corresponding definition signature cardinality and the type of entities that make up the signature can differ.

- *synonym explicit definition:* takes the form takes the form $\mathsf{A} \equiv \mathsf{B}$, where A, B are concept names. Both of these concepts are explicitly defined synonyms of each other. Σ contains a single concept name.
- *complex explicit definition:* $\mathsf{A} \equiv \mathsf{C}$ where A is an explicitly defined concept name, and C is a complex concept. C can take different forms:
  - *union:* $\mathsf{A} \equiv \mathsf{C_1} \sqcup \mathsf{C_2}$
  - *intersection:* $\mathsf{A} \equiv \mathsf{C_1} \sqcap \mathsf{C_2}$
  - *property domain or range:* $\mathsf{A} \equiv \exists \mathsf{r}.\mathsf{C}$, $\mathsf{A} \equiv \exists \mathsf{r}.\exists \mathsf{s}.\top$

  Σ is equivalent to the signature of C. If $|\Sigma| = 1$, it contains a single role name; otherwise $|\Sigma| > 1$, consisting of an arbitrary set of entities.

(2) Constituent concept of a disjoint union. A disjoint union states that a concept A is a union of a set of pairwise disjoint concepts $\{\mathsf{C_1}, \ldots, \mathsf{C_n}\}$. This is formalised by a set of axioms, therefore $|\mathcal{J}| > 1$. $\mathcal{J}$ contains a single concept equivalence axiom α_1, which states the union, as the explicit definition of A; then the corresponding axioms $\alpha_2 - \alpha_m$ state pairwise disjointness of the other signature entities.

$$\begin{aligned}\mathcal{J} = \{&\alpha_1 : \mathsf{A} \equiv \mathsf{C_1} \sqcup \ldots \sqcup \mathsf{C_n},\\ &\alpha_2 : \mathsf{C_i} \sqsubseteq \neg \mathsf{C_1},\\ &\ldots\\ &\alpha_m : \mathsf{C_i} \sqsubseteq \neg \mathsf{C_n}\}\end{aligned}$$

In this pattern, *all named and anonymous* [12] *concepts* are defined. Please note that C is not necessarily a concept name, but a potentially complex concept. A is the only explicitly defined concept (LHS of axiom α_1). Its constituent concepts $\{\mathsf{C_1}, \ldots, \mathsf{C_n}\}$ (the signature of the RHS of α_1), are all implicitly defined. To define A, all those entities which occur on the RHS of the CE axiom are needed, thus the signature of this concept definition is $\Sigma^{\mathsf{A}} = \mathsf{Sig}(\alpha_1) \setminus \{\mathsf{A}\}$.

$$\Sigma^{\mathsf{C}_j} = \{ \underbrace{\mathsf{A},}_{\text{subsumer concept}} \quad \underbrace{\mathsf{C_1}, \ldots \mathsf{C_i}, \mathsf{C_k}, \ldots, \mathsf{C_n}}_{\text{disjoint concepts}}\}$$

To define any other concept C_j (of RHS concepts) the signature would be $\Sigma^{\mathsf{C}_j} = \mathsf{Sig}(\alpha_1) \setminus \{\mathsf{C_j}\}$, where the definition axiom takes the form of C_j being defined as the conjunction of A, and the complement of the union of those concepts that C_j is disjoint with:

$$\mathsf{C_j} \equiv \mathsf{A} \sqcap \neg(\mathsf{C_1} \sqcup \ldots \sqcup \mathsf{C_i} \sqcup \mathsf{C_k} \sqcup \ldots \sqcup \mathsf{C_n}) \quad \text{where } (1 \leq i < j < k \leq n)$$

This pattern does not apply to simple unions, i.e. cases where there is a CE axiom such that $\mathsf{A} \equiv \mathsf{C_1} \sqcup \ldots \sqcup \mathsf{C_n}$, but there are no corresponding disjoint statements. Although it is true that any $\mathsf{C_j} \sqsubseteq \mathsf{A}$, but it cannot be stated that $\mathsf{C_j} \sqsubseteq \neg(\mathsf{C_1} \sqcup \ldots \sqcup \mathsf{C_i} \sqcup \mathsf{C_k} \sqcup \ldots \sqcup \mathsf{C_n})$, hence no precise definition can be constructed which would define the necessary and sufficient conditions for being a C_j.

Example 3 [13] Regular_contribution *is explicitly defined by* α_1*, where the definition axiom signature is* $\Sigma = \{\mathsf{Extended\_abstract}, \mathsf{Paper}\}$*. On the RHS of* α_1 *there are two concept names, these are declared to be pairwise disjoint in axiom* α_2*.*

$$\begin{aligned}\mathcal{J} = \{&\alpha_1 : \mathsf{Regular\_contribution} \equiv \mathsf{Extended\_abstract} \sqcup \mathsf{Paper},\\ &\alpha_2 : \mathsf{Extended\_abstract} \sqsubseteq \neg \mathsf{Paper}\}\end{aligned}$$

[12] Assuming that any anonymous concept that appear in α_1 are not unfolded, for example if $\mathsf{A} \equiv \mathsf{C_1} \sqcup \exists \mathsf{r}.\exists \mathsf{s}.\top$, then $\exists \mathsf{r}.\exists \mathsf{s}.\top$ is defined, but $\exists \mathsf{s}.\top$ is not.

[13] Conference corpus, Conference.owl

It is clearly visible, that both Extended_abstract and Paper are implicitly defined from $\mathcal{J}$ as follows:

$$\begin{aligned} \mathsf{Extended\_abstract} &\equiv \mathsf{Regular\_contribution} \sqcap \neg \mathsf{Paper} \\ \mathsf{Paper} &\equiv \mathsf{Regular\_contribution} \sqcap \neg \mathsf{Extended\_abstract} \end{aligned}$$

(3a) Domain or range restriction concept of a role. In this pattern $|\mathcal{J}| > 1$ and Σ contains a single role name. A role is defined as a relationship between a certain *domain* and *range* concept, thus in some cases a role name itself is sufficient to define its domain or role concept. For example the non-DL statement

$$\mathsf{hasChild}(\mathsf{Parent}, \mathsf{Human})$$

asserts that hasChild is a relation between instances of concept Parent and Child. In DL, this is formalised as the axiom set:

$$\text{domain restriction} \qquad \exists \mathsf{hasChild}.\top \sqsubseteq \mathsf{Parent} \tag{12}$$

$$\text{range restriction} \qquad \top \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Human} \tag{13}$$

(12: everything that has a child is a parent; 13: only a human can be child of something). These axioms alone do not define neither the domain, nor the role concept because they only express subsumption, but not equivalence. However, if the ontology contains the additional axiom (every parent has a child)

$$\mathsf{Parent} \sqsubseteq \exists \mathsf{hasChild}.\top \tag{14}$$

then (12, 14) together imply the statement, $\mathsf{Parent} \equiv \exists \mathsf{hasChild}.\top$, i.e. a parent is somebody who has a child. Therefore now the domain concept can be defined using the role itself. A range concept can also be defined, given that an additional axiom that specifies its meaning, is also present in the ontology.

Example 4 [14] Conference_setup *is defined by the signature* $\Sigma = \{\mathsf{includes\_topic}\}$. *The axioms of* $\mathcal{J}$ *entail the definition axiom* $\mathsf{Conference\_setup} \equiv \exists \mathsf{includes\_topic}.\top$, *i.e.* Conference_setup *is the domain of relation* includes_topic.

$$\begin{aligned} \mathcal{J} = \{\ & \alpha_1 : \mathsf{Conference\_setup} \sqsubseteq \exists \mathsf{includes\_topic}.\mathsf{Topic}, \\ & \alpha_2 : \exists \mathsf{includes\_topic}.\top \sqsubseteq \mathsf{Conference\_setup}\} \end{aligned}$$

(3b) Domain or range concept of role or inverse role. This pattern is almost identical to the normal role pattern (3a), as in this case there is also a role name $\mathsf{r}(\mathsf{C}, \mathsf{D})$ that defines a concept, either as its domain or range. However, there is always another role name, which is the inverse of the former role that defines the given concept, only here the domain becomes range and vice versa: $\mathsf{s}(\mathsf{D}, \mathsf{C})$ and $\mathsf{r} \equiv \mathsf{s}^-$. Because $\mathsf{r} \equiv \mathsf{s}^-$, the two roles are interchangeable, hence any axiom that is relevant to r is also relevant to s as well. In this pattern $|\mathcal{J}| > 1$ and Σ contains a single role name.

[14] CONFERENCE corpus, *confious.owl*

Example 5 [15] Document *is either definable as the domain of* used_by*, or as the range of* use.

$$\begin{aligned}\mathcal{J} = \{\ & \alpha_1 : \mathsf{used\_by} \equiv \mathsf{use}^-, \\ & \alpha_2 : \mathsf{Document} \sqsubseteq \exists \mathsf{used\_by}.\mathsf{Person} \\ & \alpha_3 : \top \sqsubseteq \forall \mathsf{use}.\mathsf{Document}\}\end{aligned}$$

Using the role used_by *as the concept definition signature, the axioms can be rewritten, and an explicit definition formulated as follows:*

$$\begin{aligned}\alpha_3 \ &\rightarrow \top \sqsubseteq \forall \mathsf{usedBy}^-.\mathsf{Document}\} \\ &\mathsf{Document} \equiv \exists \mathsf{used\_by}.\top\end{aligned}$$

Alternatively, the role use *can be used for the same purpose:*

$$\begin{aligned}\alpha_2 \ &\rightarrow \mathsf{Document} \sqsubseteq \exists \mathsf{use}^-.\mathsf{Person}\} \\ &\mathsf{Document} \equiv \exists \mathsf{use}^-.\top\end{aligned}$$

(3c) Explicitly defined concept with inverse role. This is a special pattern which differs from the previous role patterns. In this case there is a concept which is explicitly defined, where its definition contains one or more roles, which has an inverse. Hence $\Sigma > 1$: consists of the signature of the explicit definition, where the one of the roles is switched with its inverse. The justification consists of two axioms, an explicit definition, and the inverse role statement.

Example 6 [16] IASTED_member *is an explicitly defined concept, but it is also implicitly definable with the signature* $\Sigma = \{\mathsf{Member\_registration\_fee}, \mathsf{is\_paid\_by}\}$

$$\begin{aligned}\mathcal{J} = \{\ & \alpha_1 : \mathsf{IASTED\_member} \equiv \exists \mathsf{pay}.\mathsf{Member\_registration\_fee}, \\ & \alpha_2 : \mathsf{is\_paid\_by} \equiv \mathsf{pay}^-\}\end{aligned}$$

(4) Implicit synonyms In this pattern Σ contains a single concept name, similarly to the explicit synonym pattern, however the axiomatisation is more complex as $|\mathcal{J}| > 1$.

The simplest case is when a concept equivalence is represented as two inclusion axioms, such as $\{\mathsf{A} \sqsubseteq \mathsf{B}, \mathsf{B} \sqsubseteq \mathsf{A}\} \models \mathsf{A} \equiv \mathsf{B}$. In other cases, this may an unintended, conceptual error made by the ontology engineer, consider the following example

Example 7 [17] *Both concepts* Event, Document *are synonym terms of each other, however this may be an error, as the words are semantically unrelated.*

$$\begin{aligned}\mathcal{J} = \{\ & \alpha_1 : \mathsf{Event} \equiv \exists \mathsf{created\_by}.\mathsf{Person}, \\ & \alpha_2 : \mathsf{Document} \equiv \exists \mathsf{created\_by}.\mathsf{Person}\}\end{aligned}$$

[15] Conference corpus, *cocus.owl*
[16] Conference corpus, *iasted.owl*
[17] Conference corpus, *paperdyne.owl*

The following example shows a more complex axiomatisation:

Example 8 [18] *In this ontology,* Plenary_lecture *is a synonym of* Tutorial.

$$\begin{aligned}\mathcal{J} = \{\ & \alpha_1 : \mathsf{Plenary\_lecture} \equiv \exists \mathsf{is\_given\_by}.\mathsf{Plenary\_lecture\_speaker},\\ & \alpha_2 : \mathsf{Plenary\_lecture\_speaker} \equiv \exists \mathsf{give}.\mathsf{Tutorial}\\ & \alpha_3 : \mathsf{Tutorial} \equiv \exists \mathsf{is\_given\_by}.\mathsf{Tutorial\_speaker}\\ & \alpha_4 : \mathsf{Tutorial\_speaker} \equiv \exists \mathsf{give}.\mathsf{Plenary\_lecture}\\ & \alpha_5 : \mathsf{give} \equiv \mathsf{is\_given\_by}^{-}\}\end{aligned}$$

5.4 Complex patterns

The combination of a set of basic patterns together form a complex pattern. Each basic pattern consist of an axiom set, from which a concept definition is derivable; in a basic pattern, usually these axioms enable the definition of a single concept. In complex patterns, apart from the main concept in question, there are typically more than one defined concepts, hence the set of axioms (cardinality of justification $\mathcal{J}$) grows with the number of defined concepts that are substituted with their descriptions in the definition of the main concept.

Example 9 [19] *This pattern is the combination of two explicit definitions. The concept* Plenary_lecture *is explicitly defined by* α_1.

$$\begin{aligned}\mathcal{J} = \{\ & \alpha_1 : \mathsf{Plenary\_lecture} \equiv \exists \mathsf{is\_given\_by}.\mathsf{Plenary\_lecture\_speaker}\\ & \alpha_2 : \mathsf{Plenary\_lecture\_speaker} \equiv \exists \mathsf{give}.\mathsf{Tutorial}\}\end{aligned}$$

In addition, there is an alternative way to define the concept, because it is also implicitly defined as $\mathsf{Plenary\_lecture} \equiv \exists \mathsf{is\_given\_by}.\exists \mathsf{give}.\mathsf{Tutorial}$*, where* $\Sigma = \{\mathsf{Tutorial}, \mathsf{give}, \mathsf{is\_given\_by}\}$

Some complex patterns consist of clearly partitioned axiom subsets, where each subset corresponds to a basic pattern which defining a single concept. In other cases (such as the following example), these subsets may overlap, reducing the overall size (number of axioms) of the complex pattern.

Example 10 [20] *In this example, concept* Conference *is defined by the combination of two basic patterns: a disjoint union* (α_1, α_3) *and role domain pattern* (α_1, α_2)*. The two patterns overlap as both contain axiom* α_1

$$\begin{aligned}\mathcal{J} = \{\ & \alpha_1 : \mathsf{Conference} \sqsubseteq = 1\mathsf{hasName}.\top\\ & \alpha_2 : \exists \mathsf{hasName}.\top \sqsubseteq (\mathsf{Conference} \sqcup \mathsf{Session})\\ & \alpha_3 : \mathsf{Conference} \sqsubseteq \neg \mathsf{Session}\}\end{aligned}$$

$$\mathsf{Conference} \equiv = 1\mathsf{hasName}.\top \sqcap \neg \mathsf{Session}$$

[18] Conference corpus, *iasted.owl*
[19] Example from *iasted.owl, OAEI2014 Conference track corpus*
[20] Example from *myreview.owl, OAEI2014 Conference track corpus*

6 Empirical Evaluation

The goal of the evaluation was to investigate the definability landscape, in order to establish the *prevalence and the eisxtent of definability* over a large and diverse corpus of OWL ontologies; and to *characterise the behaviour* of proposed definability computation algorithms. All of the data and software, including ontologies, computed definition signatures, corresponding justifications, generated definition axioms, and other raw results, are available online[21].

6.1 Evaluation Corpus

Dataset selection was mostly dictated by *diversity*, meaning that the dataset must contain a wide range of ontologies of different *size* (with respect to signature, as well as axiomatisation), and *language expressivity* (ranging from simple to very expressive). In addition, it was important that documents were a *real world* ontologies depicting a *variety of domains*, with a broad range of *application areas*[22], and originating from a large number of independent *sources* (domain experts, ontology engineers, application developers, etc.) that may apply different modelling styles. Lastly, it was also important to have a sufficient number of ontologies in order to uncover any patterns or anomalies that may effect the derived conclusions.

Ontology format. There are a number of popular knowledge representation languages and corresponding ontology document formats available for creating ontologies. The ontology document format was restricted to OWL compatible documents. This choice was motivated by the following reasons:

- OWL is the official W3C recommendation, a de facto standard, hence it is widely used for both academic, and real world purposes.
- There is a plethora of accessible and freely obtainable OWL ontologies, in the form of individual files, and curated collections consisting of thousands of documents.
- OWL has very good tool support. This is important for the implementation of the experiment framework, which requires: an API for creating, manipulating and processing ontologies; ontology reasoner(s); API for module extraction; and the API for computing justifications, which is exclusive to OWL.

[21] `http://www.csc.liv.ac.uk/~dgeleta/ontodef.html`

[22] The purpose for which a particular ontology is used for impacts the ontology engineers' modelling choices, such as DL expressivity. For example, a highly expressive language is detrimental to the effectives of reasoning services, therefore one would not use such language when the aim is to perform large scale or frequent inferences; on the other hand, when the goal of the conceptualisation is the most accurate representation of some domain of interest, an expressive language is usually a better choice.

- OWL has several named profiles corresponding to different levels of DL language expressivity. This helps to partition the dataset, which is necessary for analysis.

6.1.1 Dataset Selection and Curation

The OWL ontology landscape was surveyed in [15, 16], providing a comprehensive picture about a number of important ontology collections and repositories. These papers present an ontology classification based on the purpose for which they were created for as either *in-use* or *test*, where the former category is built for use in real life applications, and the later is built for academic research. The evaluation corpus was assembled from six different collections.

- **OAEI corpus.** Each year since 2004 the Ontology Alignment Evaluation Initiative (OAEI[23]) [6] organises evaluation of ontology matching technologies using several *tracks*, where each track consists of different datasets, aiming to evaluate certain ontology matching features (instance matching, large ontology matching, interactive matching etc.). It contains a mixture of in-use, and test ontologies. In addition to a diverse set of ontologies, each track contains validated ontology alignments as well, which would also facilitate later experiments that meant to assess the application of definability in the ontology alignment domain. Several tracks were selected for evaluation:

  - **Anatomy track:** consists of 2 relative large ontologies, where one describes the human, the other describes the mouse anatomy.
  - **Large Biomedical Ontologies track:** consists of 6 large, semantically rich ontologies. These are extracts (overlapping fragments) of three well known biomedical ontologies: Foundational Model of Anatomy (FMA), SNOMED CT, and the National Cancer Institute Thesaurus (NCI).
  - **Conference track:** is a collection of 16 small ontologies that model the conference organisation domain.

- **NCBO BioPortal repository corpus.** This hand crafted, community-based repository is hosted by the National Center for Biomedical Ontology (NCBO) [17]. According to their website[24] it is the 'world's most comprehensive repository of biomedical ontologies'. As of August 2015, it contains over 440 ontologies, represented in OWL compatible formats (OWL, OBO, RDF and Proteg frames). These real world ontologies vary greatly in size and expressivity, which makes it a popular corpora for tool development and academic empirical evaluation.

- **TONES corpus.** This hand curated ontology repository is aimed to provide a comprehensive test set for OWL application development and

[23] `http://oaei.ontologymatching.org/`
[24] `http://bioportal.bioontology.org/`

empirical studies[25]. At the time of access, in December 2014, it contained 219 OWL and OBO ontologies, varing greatly in size and expressivity.

- **WebCrawl corpus.** This corpus was created by the authors of [16]. The corpus was obtained by crawling the web and collecting OWL ontologies. The dataset was curated by applying various filtering heuristics (file and domain based manual cleaning procedures, repairing some minor syntactic errors such as missing entity declarations, etc.), resulting in a very large (in comparison to the other listed corpora) set of non-trivial OWL ontologies containing 4327 documents. The general purpose of the corpus is to allow sampling based empirical evaluation leading to more representative results than when cherry-picking individual documents, or somewhat arbitrary selecting certain data sets.

Curation Process. In [15, 16] the authors describe the good practices and the pitfalls of selecting datasets for empirical evaluation; provide an overview of the popular ontology repositories; present a checklist for dataset curation, analysis and comparison; as well as outline an approach for gathering large yet interesting (i.e., non-trivial) collection of OWL documents.

[14] served as a source for obtaining a curated version of the BioPortal and WebCrawl datasets[26]). Both datasets were curated by the authors as follows: each document was *parsed using* the OWL API. Available *import closures* [27] were recursively downloaded, and merged into a single ontology document. The authors discarded any documents that could not be parsed, or had missing imports. Both of these two datasets were compressed and stored on-line as ZIP archive files; WebCrawl could not be unarchived entirely due to an unknown error, resulting in the marginal loss of some files.

The three OAEI datasets and the TONES corpus was also curated, each file was parsed using the OWLAPI, but missing import closures were not downloaded and merged. *Any file that could not be parsed or had any missing imports was discarded.*

In addition, each document of all datasets were tested for *consistency*, because as later described, inconsistency in an ontology leads to false definability results, hence all inconsistent documents were also discarded. The consistency check was performed using both Pellet and Hermit ontology reasoners; there were some minor discrepancies between the results of these two, therefore only those documents were kept which was found consistent by both reasoners. Fur-

[25] `http://rpc295.cs.man.ac.uk:8080/repository/`

[26] Available on-line `http://web.stanford.edu/~horridge/publications/2014/iswc/atomic-decomposition/data/`

[27] OWL ontologies, much like software, may be built in a modular fashion, where reusable ontological knowledge is kept physically separate (i.e. it is an ontology on its own), but can be included to create a single document through importing; top-level or foundation ontologies that describe very general concepts are often used across many different knowledge domains. An ontology which imports others is dependent on its imports, thus any imports must be available (online or locally), otherwise the ontology cannot be loaded. To avoid scenarios where imports cannot be loaded, or to reduce loading time (especially with large documents stored online), these are often merged with the 'base' ontology.

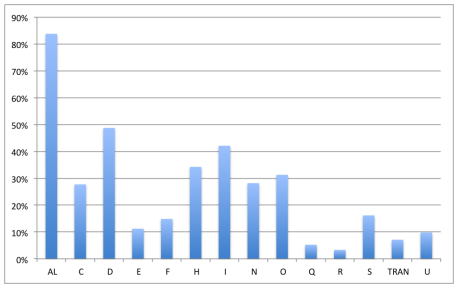

Figure 7: Relative frequency of OWL constructor usage in the corpus.

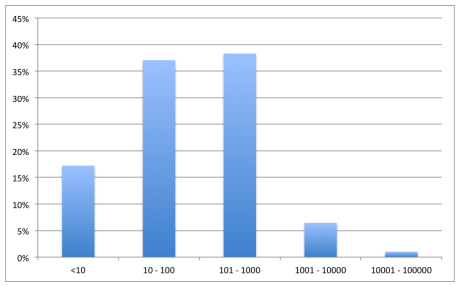

Figure 8: Distribution of ontology sizes, binned by the number of logical axioms.

thermore, in each dataset, with the exception of the OAEI corpus, some large or very expressive ontologies could not be processed (meaning that the implicit definability checking algorithm could not complete its run) due to either *reasoner time-out error*, or *memory overflow* of the Java Virtual Machine; these documents were also excluded from the final, curated versions of the corpus.

6.1.2 Dataset Description and Comparison

In order to demonstrate the diversity and to highlight certain characteristics of the evaluation corpora that may effect the prevalence and extant of concept definability, the main properties of the processed corpus are described in detail.

		Anatomy	LargeBio	Conference	TONES	BioPortal	WebCrawl
Signature	min	2755	3725	32	1	6	2
	avg	3034	18187	113	1451	2450	119
	med	3034	11809	109	168	569	53
	max	3313	51181	274	36090	45091	2744
Classes	min	2743	3696	14	0	0	1
	avg	3024	18140	54	1239	2060	61
	med	3024	11785	49	88	451	15
	max	3304	51128	140	36076	38738	2329
Object properties	min	2	0	13	0	0	0
	avg	3	36	33	33	39	20
	med	3	35	33	5	11	5
	max	3	82	58	922	1390	607
Data properties	min	0	24	0	0	0	0
	avg	0	8	12	14	9	9
	med	0	0	11	0	0	1
	max	0	24	23	708	488	674
Individuals	min	0	0	0	0	0	0
	avg	0	0	10	40	341	19
	med	0	0	0	0	0	1
	max	0	0	114	3542	22534	1014
Logical Axioms	min	4838	3828	65	0	3	0
	avg	8192	24150	265	1882	4156	215
	med	8192	15268	233	187	839	69
	max	11545	71042	739	42656	77700	4556

Table 1: Entity usage (minimum, average, median, maximum) in the six collections

	Anatomy	LargeBio	Conference	TONES	WebCrawl	Total
Full	0.0%	0.0%	0.0%	16.7%	41.2%	**42.2%**
DL	100.0%	100.0%	100.0%	83.3%	58.8%	**57.8%**
EL only	0.0%	33.3%	0.0%	10.6%	1.9%	**1.9%**
EL total	0.0%	33.3%	0.0%	31.1%	6.9%	**7.6%**
QL only	0.0%	0.0%	0.0%	0.0%	1.0%	**1.1%**
QL total	0.0%	0.0%	0.0%	20.0%	5.2%	**5.6%**
RL only	0.0%	0.0%	6.7%	2.2%	5.0%	**7.5%**
RL total	0.0%	0.0%	6.7%	13.9%	9.3%	**13.4%**
DL only	0.0%	66.7%	93.3%	49.4%	45.1%	**43.8%**

Table 2: OWL 2 profile distribution in the collections

Table 1 shows the signature, **entity usage** (break down of classes, properties, individuals) and the number of logical axioms in the six datasets. In addition, Figure 8 shows the distribution of ontology sizes sorted into six size bins (ranging from less than 10 to 100000 axioms) by the number of logical axioms [28]. In terms of logical axiom count, the main corpus (i.e., the union of the six collections) is sufficiently diverse in size: the majority falls into the *very small* (less than 10 axioms) and *small* (10 to 100 axioms) size bins, with 17% and 37%, respectively; *medium* size (101 to 1000 axioms) ontologies constitute to 38% of the dataset; about 6% is *large* (1001 to 10000 axioms), and 1% falls into the *very large* (over 10000 axioms) size category, making up the remainder of the corpora. Although these numbers are the result of measuring the asserted ontology model, which could obviously differ from the inferred model of an ontology [29], the mean and median values of the signature and entities confirm the overall size distribution.

The **expressivity** of a particular DL flavour depends on the combination of the concept and role constructors used by the language. Within this corpus, there are over 250 different constructor combinations, thus the dataset is sufficiently diverse. Although the constructor combinations are not listed here (for the sake of readability), the use of constructors is depicted in Figure 7, where the frequency of constructor usage is shown in terms relative usage. Constructor combinations fall within OWL2 profiles, where each profile is a syntactic subset of the full OWL2 language. These profiles were created to facilitate certain purposes, trading expressive power for efficient reasoning.

Table 2 shows the named profiles in the corpus (for each sub-collection, as well as the whole). Please note that an ontology can fall within more than one named profile. In addition to the three named profiles, there are two more categories, 'DL' shows any ontology that do not belong to any named profiles, but are a valid syntactic subset of the OWL 2 language; 'Full' denotes those OWL documents that cannot be classified as 'DL', but were still parsable by the OWL API and processable by reasoners (as noted in [16], this could occur when,

[28] A more detailed logical axiom distribution that shows the breakdown for each corpus can be found in the Appendix **??**, Figure **??** (absolute distribution and relative distribution).

[29] In some cases, the asserted model of an ontology may contain less terminological axioms than the inferred model, even though these models are semantically equivalent. For example, the asserted model consisting of a single axiom $\mathsf{A} \equiv \mathsf{B} \sqcup \mathsf{C}$ conveys the same meaning as the inferred model, i.e. the set of axioms $\{\mathsf{A} \equiv \mathsf{B} \sqcup \mathsf{C}, \mathsf{B} \sqsubseteq \mathsf{A}, \mathsf{C} \sqsubseteq \mathsf{A}\}$.

Axiom type	Anatomy	LargeBio	Conference	TONES	BioPortal	WebCrawl
EquivalentClasses	0.0%	34.6%	66.7%	39.3%	45.6%	38.1%
SubClassOf	100.0%	100.0%	100.0%	97.0%	87.5%	76.2%
DisjointClasses	50.0%	88.5%	66.7%	53.5%	42.7%	35.1%
DisjointUnion	0.0%	0.0%	0.0%	0.0%	0.5%	0.0%
ClassAssertion	100.0%	23.1%	0.0%	30.8%	22.4%	51.2%
EquivalentObjectProperty	0.0%	0.0%	0.0%	5.5%	2.2%	2.2%
SubObjectPropertyOf	0.0%	36.5%	50.0%	50.6%	34.0%	33.2%
DisjointObjectProperty	0.0%	0.0%	0.0%	3.2%	0.4%	0.2%
ObjectPropertyDomain	0.0%	100.0%	33.3%	51.7%	42.3%	54.6%
ObjectPropertyRange	0.0%	100.0%	33.3%	49.0%	43.0%	56.3%
ObjectPropertyAssertion	0.0%	0.0%	0.0%	11.0%	8.7%	16.5%
NegativeObjectPropertyAssertion	0.0%	0.0%	0.0%	0.0%	0.0%	0.9%
InverseObjectProperties	0.0%	100.0%	0.0%	45.4%	28.3%	38.3%
TransitiveObjectProperty	50.0%	32.7%	0.0%	46.7%	27.4%	21.5%
SymmetricObjectProperty	0.0%	7.7%	0.0%	18.5%	12.9%	11.4%
AsymmetricObjectProperty	0.0%	0.0%	0.0%	1.6%	1.0%	0.7%
FunctionalObjectProperty	0.0%	63.5%	0.0%	32.4%	23.7%	22.8%
InverseObjectProperty	0.0%	51.9%	0.0%	7.9%	8.4%	9.2%
IrreflexiveObjectProperty	0.0%	0.0%	0.0%	3.8%	0.4%	0.7%
EquivalentDataProperty	0.0%	0.0%	0.0%	1.1%	0.5%	1.7%
SubDataPropertyOf	0.0%	3.8%	0.0%	12.0%	1.4%	10.4%
DisjointDataProperty	0.0%	0.0%	0.0%	0.0%	0.0%	0.1%
DataPropertyDomain	0.0%	71.2%	33.3%	26.5%	21.3%	37.5%
DataPropertyRange	0.0%	71.2%	33.3%	28.0%	25.3%	38.7%
FunctionalDataProperty	0.0%	59.6%	33.3%	19.9%	16.8%	19.9%
DataPropertyAssertion	0.0%	0.0%	0.0%	11.9%	4.0%	13.5%
SameIndividual	0.0%	0.0%	0.0%	0.6%	1.1%	17.4%
DifferentIndividuals	0.0%	3.8%	0.0%	11.8%	3.2%	6.0%
SubPropertyChainOf	0.0%	0.0%	33.3%	9.6%	2.2%	1.6%

Table 3: Axiom type usage in the six collections, as a proportion of ontologies that use an axiom type

for instance, entity declarations are missing from the document). A small part (16.7%) of the BioPortal corpus, and a large portion (41.2%) of WebCrawl were outside of the DL profile, the rest of the documents were all within the DL profile.

The OWL language categorises represented axioms into **axiom types**, for example a concept synonym statement such as $\mathsf{C} \equiv \mathsf{D}$ is formalised by the *EquivalentClasses* axiom type. Table 3 shows the axiom type usage in each subsets of the evaluation corpus, which, due to the above mentioned problem of OWL profiling provides a more fine grade picture about the expressivity of the ontologies than the OWL 2 profiles.

6.2 Results

The **experimental framework** was implemented in Java; the OWL API version 4.1.0 was used for ontology manipulation and for interacting with the reasoners [13]. The OWL Explanation API[30] was usd to compute justifications. Both the HermiT [9] (version 1.3.8) and Pellet [22] (version 2.2.2) reasoner[31]. Experiments were conducted with a 10 minute reasoner timeout setting, and 4GB maximum memory allocated for the Java Virtual Machine, on an average

[30] `https://github.com/matthewhorridge/owlexplanation`

[31] In most datasets HermiT performs faster, however Pellet was able to load and process some ontologies that HermiT could not (due to ontologies using datatypes that are not part of the OWL 2 datatype map and no custom datatype definition was given).

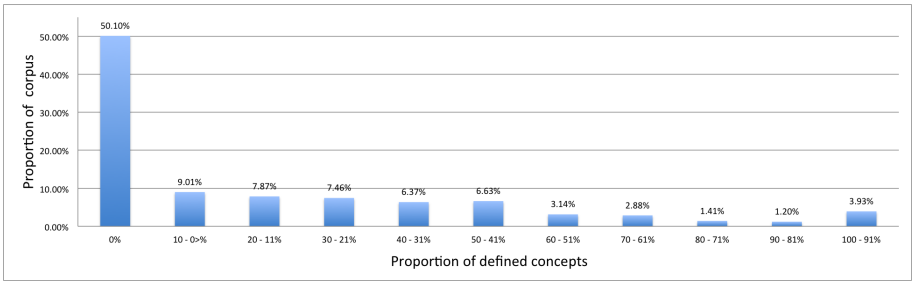

Figure 9: Proportion of a ontologies in the corpus, binned by the proportion of defined concepts in an ontology

configuration machine with 16GB RAM and a 4 core processor architecture. The **evaluation corpus** was assembled from a variety of OWL ontology datasets, including ontologies in the Manchester Ontology Repository [32] and those used for the OEAI evaluation challenge [33]. The corpus contains 3576 ontologies of different *size* (w.r.t. signature cardinality, as well as axiomatisation), and *language expressivity*. The ontologies contain 396943 concepts in total; all the datasets used are described in [8].

We distinguish between *defined* and *undefined* ontologies depending on whether they contain at least one defined concept, of if all their concepts are undefined. The investigation into the prevalence and the extent of definability was motivated by the following **experimental questions**: (1) what portion of ontologies contain defined concepts; (2) what is the ratio of undefined and defined concepts in defined ontologies; (3) what is the ratio of explicitly and implicitly defined concepts in defined ontologies; (4) what ontology properties affect definability? In order to answer the first three questions, all concepts in every ontology were examined to decide whether they were undefined, or defined either explicitly or implicitly. In general about half (47.62%), 1703 out of 3576 ontologies contained *at least one* defined concept. Out of all concepts (396943 concepts), 75.82% were undefined, 20.74% were explicitly and 3.44% were implicitly defined. Figure 9 shows the proportion of documents in the corpus, binned by the ratio of defined to undefined concepts within an ontology. As expected, there are increasingly fewer number of ontologies as the ratio of defined to undefined concepts increase.

Intuitively concept definability is mostly influenced by the *axiomatic richness* of an ontology (i.e. the granularity of conceptualisation), along with a combination of indirect factors, such as the *expressivity* of the employed DL language and the *size* of an ontology. Other properties such as origin (source of creation), and the conceptualised domain of interest are less likely to have significant effect on definability. In the following, each ontology property is assessed, by comparing the defined and undefined parts of the corpus.

[32] `http://owl.cs.manchester.ac.uk/tools/repositories/`
[33] `http://oaei.ontologymatching.org`

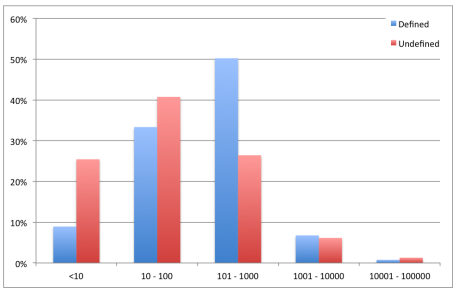

Figure 10: Relative distribution of defined and undefined ontology sizes, binned by the number of logical axioms.

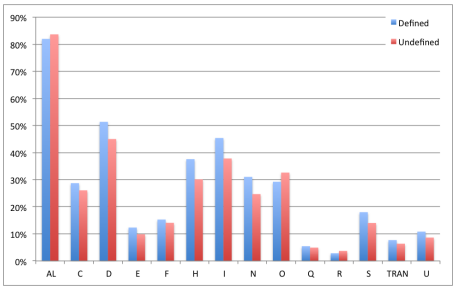

Figure 11: Comparing the frequency of OWL constructor usage in defined and undefined ontologies.

Logical Axioms. Figure 10 confirms the claim that size is at best an indirect influencing factor of concept definability. The last two (largest) size bins roughly have the same number of ontologies in defined and undefined categories as well. In the smallest size bin (≤ 10) there are significantly more undefined ontologies, in the medium size bin ($101 - 1000$) there are twice as many defined ontologies than undefined ones. This is most likely because an ontology that contains over a hundred concepts: (a) is more likely to contain specialised and generalised concepts that are distinguished in their meaning by more descriptive axioms; (b) takes considerable effort to develop, which could also translate into making the ontology more functional by adding more content (axioms); (c) the developer has deeper understanding over the subject domain, which is also reflected by richer axiomatisation of the described knowledge.

Ontology	$\mathcal{EL}$ only	$\mathcal{EL}$	$\mathcal{QL}$ only	$\mathcal{QL}$	$\mathcal{RL}$ only	$\mathcal{RL}$	$\mathcal{DL}$ only	$\mathcal{DL}$	Full
Defined	2%	6%	1%	4%	4%	8%	49%	61%	39%
Undefined	2%	8%	1%	6%	6%	11%	41%	56%	44%

Table 4: OWL 2 profiles contrasting ontologies with and without defined concepts

OWL Profiles and DL Constructors. Table 4 shows the distribution of OWL profiles in defined and undefined ontologies. Profiling results are inconclusive, however, there is a slight indication that the more expressive ontologies are more likely to contain defined concepts than less expressive ones. The *DL only* category (ontologies exclusively in the DL profile) is used by 49% of the defined ontologies, whereas it is only used in 41% of the undefined ones. Figure 11 shows the constructor usage in defined and undefined ontologies; this shows an even distribution, meaning that both categories contain less and more expressive ontologies.

MDSs. Figure 15 shows the number of MDSs of all defined concepts found in the corpus, divided into two parts corresponding to the search and expan-

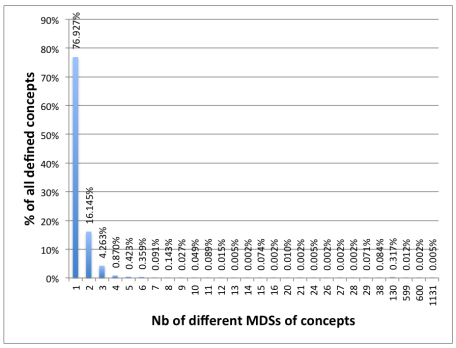

Figure 12: Relative distribution of MDSs, binned by the number of different MDSs per concept.

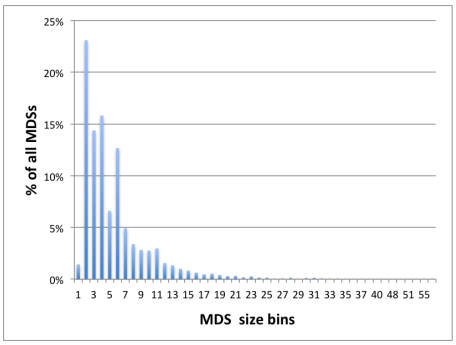

Figure 13: Relative distribution of MDS cardinality, binned by the number of entities in an MDS

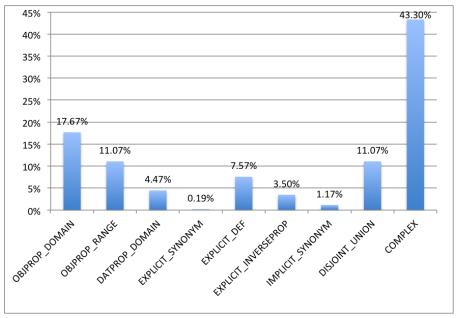

Figure 14: Relative distribution of MDSs binned by concept definition pattern types

Type of Definability		MCDS approach DISJOINT	EXPANSION
Explicitly	min	1	1
	avg	1.76	3.97
	med	1	2
Implicitly	min	1	1
	avg	1.96	2.34
	med	1	1

Figure 15: Measuring the number of MDSs of explicitly and implicitly defined concepts

sion phases of MDS computation. The incomplete, but polynomial time *search* phase computes pairwise disjoint MDSs; the complete, but exponential time *expansion* phase includes computing the complete set of MDSs of defined entities. The results are divided between explicitly and implicitly defined concepts. A given defined concept has on average 3.06 different MDSs. The median score of 2 suggests that at least half of all concepts had two or more different definitions. This is supported by Figure 15, which provides the distribution of MDSs, binned by the number of different MDSs per concept: 34.56% of all defined concepts had exactly one MDS; the remaining 65.44% of defined concepts had at least two different MDSs, where 22.52% of concepts had exactly two MDS. The trend is that as the number of MDSs per same concept is increasing, the number of defined concepts decreases. By comparing the mean scores of MDSs per concepts between the two definability types, the data suggests that, in general explicitly defined concepts have more MDSs than implicitly defined ones; the former has 3.94, the later has 2.34 different MDSs on average. This is most likely because, by definition, an explicitly defined concept is guaranteed to have at least one, but potentially more MDSs (unless the explicit definition is cyclic). The rest of

the concept's MDSs correspond to implicit definitions of the concept. Figure 6 shows that most concepts had only a few different MDSs, however few concepts had a large number of MDSs, which is explained by that the number of MDSs is potentially exponential in the size of the ontology.

MDS were analysed according to two properties: (a) *cardinality*; (b) the *corresponding CDP*. MDS cardinality is one of the indicators of its corresponding CDP type, and as such this property is used in pattern recognition. MDSs are divided into two size categories: *single*, and *multi-entity signatures*. Figure 13 presents the distribution of MDS sizes, binned by the number of entities in an MDS. 34.56% of all MDSs contained only a single entity, the remainder contained two or more entities, the maximum number of entities in a signature was seven (in 1.55% of all MDSs). In general as the size of the MDS increases, the occurrence of the size type decreases. Figure 14 shows the distribution of MDSs binned by CDP types. Out of all CPDs, 43.3% were complex. The rest of the CDPs fall into the basic pattern category, for which it is possible to generate a definition axiom, using the presented approach. 10.49% of all defined concepts had only complex CDPs, the majority either had only basic pattern MDSs, or both at the same time.

Modelling errors. Out of all defined concepts, 15% has redundant concepts, and only a handful of concepts were definable by an empty signature. Interestingly, most of these errors were detected in highly used and extensively curated ontologies, confirming the intuition that definability can aid ontology engineering. This result also suggests, that redundant concepts can occur because knowledge engineers aim to make explicit definitions descriptive or more comprehensible for humans rather than succinct.

7 Conclusions and Future Work

In this paper we have presented a novel way to compute the complete set of definition signatures of defined concepts and roles. A large and diverse set of ontologies were subjected to definability computation, which included establishing the definability status of concepts and computing all possible minimal definition signatures of defined concepts. In addition it was shown, that definability computation is feasible for most real world ontologies, and in some cases, it can be useful in dynamic environments as well, due to the fact that a subset of MDSs can be found in polynomial time. This has confirmed the hypothesis that definability is prevalent in any type of ontology, although it is more likely to occur in more expressive, and semantically richer ontologies. Hence the exploitation of MDSs could indeed benefit several application areas. One application area, the support of ontology engineering by detecting modelling errors using MDSs was introduced in this paper. In terms of future work, we aim to explore other possible applications of definability such as ontology alignment evaluation evaluation, and ontology alignment negotiation. These new applications should be investigated in depth in order to quantify, the intuitively significant, effect

of rewriting within these contexts.

References

[1] Franz Baader. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, 2003.

[2] Evert W Beth. On padoa's method in the theory of definition. *Indagationes Mathematicae*, 15:330 – 339, 1953.

[3] Michelle Cheatham, Zlatan Dragisic, Jérôme Euzenat, Daniel Faria, Alfio Ferrara, Giorgos Flouris, Irini Fundulaki, Roger Granada, Valentina Ivanova, Ernesto Jiménez-Ruiz, Patrick Lambrix, Stefano Montanelli, Catia Pesquita, Tzanina Saveta, Pavel Shvaiko, Alessandro Solimando, Cássia Trojahn dos Santos, and Ondrej Zamazal. Results of the ontology alignment evaluation initiative 2015. In *Proceedings of the 10th International Workshop on Ontology Matching*, pages 60–115, 2015.

[4] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[5] Chiara Del Vescovo, Pavel Klinov, Bijan Parsia, Ulrike Sattler, Thomas Schneider, and Dmitry Tsarkov. Empirical study of logic-based modules: Cheap is cheerful. In *The Semantic Web–ISWC 2013*, pages 84–100. Springer, 2013.

[6] Jérôme Euzenat, Christian Meilicke, Heiner Stuckenschmidt, Pavel Shvaiko, and Cássia Trojahn. Ontology alignment evaluation initiative: Six years of experience. In Stefano Spaccapietra, editor, *Journal on Data Semantics XV*, pages 158–192. Springer-Verlag, Berlin, Heidelberg, 2011.

[7] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching, Second Edition*. Springer, 2013.

[8] David Geleta, Terry R. Payne, Valentina Tamma, and Frank Wolter. Computing minimal definition signatures in description logic ontologies. Technical report, University of Liverpool, 2016.

[9] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: an owl 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.

[10] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. Owl 2: The next step for owl. *Web Semant.*, 6(4):309–322, November 2008.

[11] Eva Hoogland et al. *Definability and interpolation: Model-theoretic investigations*. Institute for Logic, Language and Computation, 2001.

[12] Matthew Horridge. *Justification Based Explanation in Ontologies*. PhD thesis, University of Manchester, 2011.

[13] Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semantic Web*, 2(1):11–21, 2011.

[14] Matthew Horridge, Jonathan M. Mortensen, Bijan Parsia, Ulrike Sattler, and Mark A. Musen. A study on the atomic decomposition of ontologies. In *Proceedings of the 13th International Semantic Web Conference - Part II*, ISWC '14, pages 65–80, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[15] Nicolas Matentzoglu, Samantha Bail, and Bijan Parsia. A corpus of owl dl ontologies. In *Proc. DL'13*, 2013.

[16] Nicolas Matentzoglu, Samantha Bail, and Bijan Parsia. A snapshot of the owl web. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, JosianeXavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web – ISWC 2013*, volume 8218 of *Lecture Notes in Computer Science*, pages 331–346. Springer Berlin Heidelberg, 2013.

[17] Natalya F Noy, Nigam H Shah, Patricia L Whetzel, Benjamin Dai, Michael Dorf, Nicholas Griffith, Clement Jonquet, Daniel L Rubin, Margaret-Anne Storey, Christopher G Chute, et al. Bioportal: ontologies and integrated data resources at the click of a mouse. *Nucleic acids research*, page gkp440, 2009.

[18] Dominique Ritze, Johanna Völker, Christian Meilicke, and Ondrej Šváb-Zamazal. Linguistic analysis for complex ontology matching. In *Proceedings of the 5th International Conference on Ontology Matching*, volume 689, pages 1 –12. CEUR-WS. org, 2010.

[19] Ulrike Sattler, Thomas Schneider, and Michael Zakharyaschev. Which kind of module should i extract? *Description Logics*, 477:78, 2009.

[20] François Scharffe, Ondřej Zamazal, and Dieter Fensel. Ontology alignment design patterns. *Knowledge and information systems*, 40(1):1–28, 2014.

[21] Inanç Seylan, Enrico Franconi, and Jos De Bruijn. Effective query rewriting with ontologies over dboxes. In *IJCAI*, volume 9, pages 923–929. Citeseer, 2009.

[22] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007.

[23] Heiner Stuckenschmidt, Livia Predoiu, and Christian Meilicke. Learning complex ontology alignments a challenge for ilp research. In *Proceedings of the 18th International Conference on Inductive Logic Programming*, 2008.

[24] Ondřej Šváb-Zamazal, Vojtěch Svátek, and Luigi Iannone. Pattern-based ontology transformation service exploiting oppl and owl-api. In *Knowledge Engineering and Management by the Masses*, pages 105–119. Springer, 2010.

[25] Balder Ten Cate, Willem Conradie, Maarten Marx, and Yde Venema. Definitorially complete description logics. *KR*, 6:79–89, 2006.

[26] Balder Ten Cate, Enrico Franconi, and Inanç Seylan. Beth definability in expressive description logics. *J. Artif. Intell. Res.(JAIR)*, 48:347–414, 2013.

[27] Brian Walshe, Rob Brennan, and Declan O'Sullivan. A comparison of complex correspondence detection techniques. volume 946, 2012.