Graph-Based Ontology Checking *

Sik Chun (Joey) Lam
Department of Computing Science
University of Aberdeen
Aberdeen, Scotland, UK
slam@csd.abdn.ac.uk

Derek Sleeman
Department of Computing Science
University of Aberdeen
Aberdeen, Scotland, UK
sleeman@csd.abdn.ac.uk

Wamberto Vasconcelos
Department of Computing Science
University of Aberdeen
Aberdeen, Scotland, UK
wvasconc@csd.abdn.ac.uk

ABSTRACT

Despite the growing availability of ontologies, the reuse of existing ontologies is often not possible without considerable effort. When one wants to reuse an ontology by importing it into ontology editors, the import process is not always successful due to an ill-formed content. Many existing ontology editors provide consistency detection by connecting to a reasoner, and highlighting the inconsistencies. However, no explanation nor functionalities are provided for users to correct the problems. In this paper we introduce a graph-based approach to checking ontology inconsistencies, implemented in our tool, ReTAX++. The system not only highlights the inconsistencies in an ontology by interacting with a reasoner, but also provides facilities for users to resolve the problems. By formalising an ontology as a graph, we check which relationships of the inconsistent concepts may cause the contradiction, and a number of options to resolve the problems are provided. Currently, we only focus on disjointedness and complement contradictions.

Categories and Subject Descriptors

I.2.4 [**Knowledge Representation Formalisms and Methods**]: representation language

General Terms

Knowledge Management

Keywords

Ontology Consistency

1. INTRODUCTION

Reusing an ontology is far from an automated process, instead it requires a significant effort from the knowledge engineer [4]. Published ontologies do not have guaranteed error-free content, that is, the concept definitions, relationships and formal axioms may contain errors. McGuinness et al. [10] found extensive need for ontological reorganisation. It is crucial to evaluate imported ontologies before re-using them in Semantic Web applications. There are many ontology tools capable of importing existing ontologies, that can work with ontologies in different formalisms [9]. However, they only indicate the inconsistent concepts, and no explanation nor functionalities are provided for engineers to correct the problems. Therefore, there is a growing need for tools that can validate ontological consistency and provide guidance as to how to correct the detected errors [2].

In this paper we propose a graph-based approach implemented in our tool, ReTAX++, to help knowledge engineers resolve inconsistencies in ontologies. The system highlights the inconsistent concepts by connecting to an external reasoner, which detects the inconsistencies in an ontology. We then check which relationships among concepts cause the problems using graph-based algorithms. An ontology is first mapped onto a graph whose nodes correspond to "units" of information and arcs to inferential dependencies between these units. The contradictions of concepts are then identified by analysing the paths of a graph. After identifying the relationships among concepts which cause the contradiction, we then provide a number of options for the user to resolve the problems. Once the user selects an option, the changes on the ontology are carried out automatically. Inconsistencies in an ontology may be caused by cardinalities, relation restrictions, concept axioms, etc.; in this paper, we only focus on the disjointedness and complement of concepts.

This paper is structured as follows. The preliminary definitions are presented in section 2. Section 3 describes the process of detecting and resolving the disjoint and complement contradictions. The system implementation is described in section 4. Section 5 is a

brief overview and a comparison of related work. Finally, we conclude in Section 6, discussing our proposal and future work.

2. PRELIMINARY DEFINITIONS

We shall formally represent an ontology as a directed graph with a finite set of nodes and edges. There are three types of nodes, *viz.* the concepts and relations of an ontology are represented as nodes in the graph; the intersection, union, complement and disjointedness operations between concepts are represented as $\sqcap$, $\sqcup$, $\neg$ and $\otimes$ nodes respectively. The nodes are linked by edges.

DEFINITION 1. *An ontology is a pair* $O = \langle \mathbf{N}, \mathbf{E} \rangle$ *where*

- $\mathbf{N}$ *is a finite and non-empty set of nodes, consisting of concepts, relations and the concept operators. We differentiate subsets of* $\mathbf{N}$*, that is* $\mathbf{N} = \mathbf{C} \cup \mathbf{R} \cup \mathcal{A} \cup \mathbf{I} \cup \mathbf{P}$*, where*
  - $\mathbf{C}$ *is a finite and non-empty set of concepts,* $\mathbf{C} = \{C_1, C_2, \ldots C_n\}$*, each* C_i *is defined below;*
  - $\mathbf{R}$ *is a finite or possibly empty set of binary quantified relations between concepts, that is each* $R \in \mathbf{R}$ *expresses relationships between concepts with quantified restriction;*
  - $\mathcal{A}$ *is a finite or possibly empty set of attributes which are the characteristic features describing concepts,* $\mathcal{A} = \{A_1, A_2, \ldots A_n\}$*, each* A_i *is defined below;*
  - $\mathbf{I}$ *is a set of individuals, that is, its elements represent actual objects of the world;*
  - $\mathbf{P} = \{ \langle l_1, \sqcap \rangle, \langle l_2, \sqcup \rangle, \langle l_3, \neg \rangle, \langle l_4, \otimes \rangle \}$ *represents the intersection, union, complement and disjointedness of concepts, and* l_1, l_2, l_3, l_4 *are the node labels.*
- $\mathbf{E} : \mathbf{N} \times \mathbf{L} \times \mathbf{N}$ *is a finite and possibly empty set of labeled edges,* $\mathbf{L}$ *is a set of labels.*

DEFINITION 2. *A concept* C *is a triple* $C = \langle c, \mathbf{A}, I \rangle$*,* $C \in \mathbf{C}$*, where*

- c *is the concept name, which is written in a sans serif font* like this*;*
- $\mathbf{A}$ *is a possibly empty or finite set of attributes,* $\mathbf{A} = \{A_1, A_2, \ldots, A_n\} \subseteq \mathcal{A}$*,* A_i *is defined below;*
- I *is a possibly empty or finite set of individuals,* $I = \{i_1, i_2, \ldots, i_n\} \subseteq \mathbf{I}$*.*

DEFINITION 3. *An attribute is a pair* $A = \langle a, \mathbf{V} \rangle$ *where* a *is the attribute name,* $\mathbf{V} = \{v_1, \ldots, v_n\}$ *is a non-empty and possibly infinite set of attribute values.*

DEFINITION 4. *A relation* $\mathbf{R} : \boldsymbol{r} \times \mathbf{N} \times \mathbf{N} \times \mathbf{Q}$ *formally establishes a quantified binary relation between two nodes, such that* $R = \langle r, N, N', Q \rangle$*, where*

- r *is the relation name, which is written in a sans serif font* like this*;*
- N *is the node participating in the relation;*
- N' *is the node linked by the relation;*
- $Q \in \mathbf{Q} = \{\exists, \forall, \ni, \leq, \geq, =\}$ *is the quantifier to restrict the number of individuals that belong to the concept* C*, they refer to existential, universal, hasValue, minimum cardinality, maximum cardinality and exact cardinality restrictions, respectively.*

An edge is an element of $\mathbf{N} \times \mathbf{L} \times \mathbf{N}$, that is any ordered pair of nodes with a label, which is possibly nil. A label edge $e = \langle n_0, \alpha, n_1 \rangle$ is said to be incident with nodes n_0 and n_1, where α is the label of the edge, $n_0, n_1 \in \mathbf{N}$, $e \in \mathbf{E}$. The label of an edge α indicates the predefined relationships between nodes. The relationships are defined as follows:

- $\sqsubseteq : \mathbf{C} \times \mathbf{C}$ formally represents the concept-subconcept relationships for the taxonomy, such that if $C' \sqsubseteq C$, then C' is a subconcept of C;
- $\equiv : \mathbf{C} \times \mathbf{C}$ formally represents equality between concepts, such that if $C' \equiv C$ then C' is equivalent to C;
- Domain: $(\mathbf{A} \cup \mathbf{R}) \rightarrow 2^C$ gives a set of domain concepts C for all relations or attributes $X \in (\mathbf{A} \cup \mathbf{R})$;
- Range: $\mathbf{R} \rightarrow 2^C$ gives a set of range concepts C for all relations $R \in \mathbf{R}$.

Paths in a graph, represented as Π, are alternating sequences of nodes and edges such that each edge in the sequence is preceded by its source node. More than one edge cannot exist between two nodes, therefore, an edge in a graph can always be determined by its source and target nodes, for brevity, a path is abbreviated by just enumerating the nodes $\langle n_0, n_1, \ldots, n_m \rangle$. The concatenation of paths is denoted by "$\cdot$". For example, $\langle n \rangle \cdot \langle m \rangle = \langle n, m \rangle$. The length of a path, $|\Pi|$, is the number of nodes in the sequence.

DEFINITION 5. *A path* Π *can be defined recursively as:*

- $\langle\rangle$ *is a path, called the empty path;*
- $\langle n\rangle$ *is a path if and only if* $n \in \mathbf{N}$*;*
- $\Pi \cdot \langle n, m\rangle$ *is a path if and only if* $\langle n, \alpha, m\rangle \in \mathbf{E}$ *and* $\Pi \cdot \langle n\rangle$*.*

DEFINITION 6. Π' *is called the common-subpath of* Π_1 *and* Π_2 *if and only if* $\Pi_1 = \Pi' \cdot \Pi_3$ *and* $\Pi_2 = \Pi' \cdot \Pi_4$*, that is two paths are decomposed into two parts, and their first parts have the maximum common subpath, where* $|\Pi'| > 0$*,* $|\Pi_3| > 0$*,* $|\Pi_4| > 0$ *and* Π' *is the maximum length path that satisfies this.*

DEFINITION 7. *The predicate matched-tail*$(\Pi_1,\ \Pi_2)$ *is true if and only if* $\Pi_1 = \Pi' \cdot \Pi_2$*,* $|\Pi'| > 0$*.*

We now explain how a set of concept paths is obtained with a given concept and an ontology. In the *concept-path* function (Figure 1), the concept C, ontology O and a empty set of paths Φ are initialised. For each relation R of C, a path Π containing C and R is created (cf. line 5). The node N linked by R, the path Π, Φ and O are input into the function *create-path* (cf. line 6).

In the function *create-path* (Figure 2), the path Π is appended with node N (cf. line 2). Three cases are checked: (1) if N is a concept, Π is added to Φ and the function ends (cf. line 4). (2) if N is an intersection, union or complement operation node, every node which is linked by N is input to *create-path* recursively (cf. line 5-8) (Note that an intersection or union node can link to more than one concept). (3) if N is a relation, the node linked by N is input to *create-path* recursively (cf. line 9-10).

As the relations of a concept could be inherited from its superconcepts or equivalent concepts, paths are also included from these concepts. That is, a path is created for each relation of its superconcepts or equivalent concepts (cf. line 8-11 in Figure 1). Similarly, the function *disjoint-path* (Figure 3) returns a set of disjoint paths with a given disjointedness node N. All nodes linked by N are input to *create-path* (cf. line 4-6 in Figure 3).

For example, in Figure 4, the concept Lion is defined so that its individuals have an eats relationship linked to at least an individual which is both a member of Cow and Sheep. This is represented as a graph with five nodes, which are C, R, C_1, C_2, N respectively, and four edges. The set of paths Φ is $\{\langle C, R, N, C_1\rangle, \langle C, R, N, C_2\rangle\}$.

¿From the *concept-path* function, we notice that a concept path always ends with a concept node. As the axioms of a concept are inherited by its subconcepts, the concept node at the end of a path Π must be substituted by its superconcepts in order to include the

1. Function *concept-path*(C, Φ, O)
2. $C = \langle c, \mathbf{A}, I\rangle$; $O = \langle \mathbf{N}, \mathbf{E}\rangle$;
3. $\Phi := \{\}$;
4. for each $R = \langle r, C, N, Q\rangle \in \mathbf{N}$
5. $\Pi := \langle C, R\rangle$;
6. *create-path*(N, Π, Φ, O);
7. end for
8. for each $\langle C, \sqsubseteq, C'\rangle$, $\langle C, \equiv, C'\rangle \in \mathbf{E}$
9. for each $R = \langle r, C', N', Q\rangle \in \mathbf{N}$
10. $\Pi := \langle C, C', R\rangle$;
11. *create-path*(N', Π, Φ, O);
12. end for
13. end for
14. End function

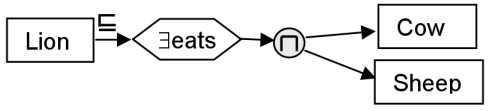
Figure 1: The concept-path function

1. Function *create-path*(N, Π, Φ, O)
2. $\Pi := \Pi \cdot \langle N\rangle$; $O = \langle \mathbf{N}, \mathbf{E}\rangle$;
3. if $N = \langle c, \mathbf{A}, I\rangle \in \mathbf{N}$ then
4. $\Phi := \Phi \cup \{\Pi\}$;
5. else if $N \in \{\langle l_1, \sqcap\rangle, \langle l_2, \sqcup\rangle, \langle l_3, \neg\rangle\}$ then
6. for each $\langle N, \alpha, N'\rangle \in \mathbf{E}$
7. *create-path*(N', Π, Φ, O);
8. end for
9. else if $N = \langle r, N', N'', Q\rangle \in \mathbf{N}$ then
10. *create-path*(N'', Π, Φ, O);
11.end if
12.End function

Figure 2: The create-path function

1. Function *disjoint-path*(N, Φ, O)
2. $\Pi := \langle\rangle$; $\Phi := \{\}$; $O = \langle \mathbf{N}, \mathbf{E}\rangle$;
3. if $N = \langle l, \otimes\rangle \in \mathbf{N}$ then
4. for each $\langle N, \alpha, N'\rangle \in \mathbf{E}$
5. *create-path*(N', Π, Φ, O);
6. end for
7. end if
8. End function

Figure 3: The disjoint-path function



$\mathbf{N} = \{C = \langle \mathsf{Lion}, \emptyset, \emptyset\rangle,$
$C_1 = \langle \mathsf{Cow}, \emptyset, \emptyset\rangle,$
$C_2 = \langle \mathsf{Sheep}, \emptyset, \emptyset\rangle,$
$R = \langle \mathsf{eats}, C, N, \exists\rangle,$
$N = \langle n_1, \sqcap\rangle\}$
$\mathbf{E} = \{\langle C, \sqsubseteq, R\rangle, \langle R, nil, N\rangle, \langle N, nil, C_1\rangle, \langle N, nil, C_2\rangle\}$

Figure 4: lion concept

1. Function *substitute*(Π, C, Π')
2. $\Pi = \Pi_1 \cdot \langle C' \rangle$;
3. if $\exists \langle C', \sqsubseteq, C \rangle, \langle C', \equiv, C \rangle \in \mathbf{E}$
4. $\Pi' := \Pi_1 \cdot \langle C \rangle$;
5. end if
6. End function

Figure 5: The substitute function

axioms from its superconcepts. This is implemented by the function *substitute* which returns another path Π'. If Π' is contradictory with other paths, then Π will be also.

3. DISJOINT AND COMPLEMENT CONTRADICTIONS

Inconsistencies may easily occur in ontologies [12], cardinality, relation restrictions and concept axioms etc can result in problems. In this paper, we analyse the relationships of concepts and concept axioms based on graph theory, and only focus on disjointedness and complement of concepts.

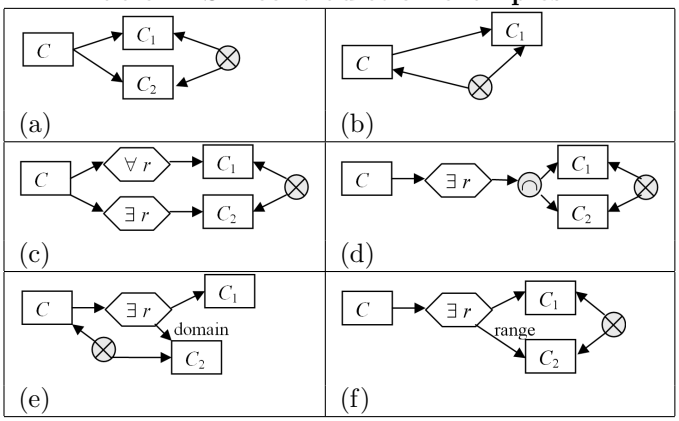
Table 1 shows some common contradictions existing in ontologies.
(a) a concept has two or more superconcepts which are disjoint with or complement of each other.
(b) a concept is disjoint with or complement of its superconcepts.
(c) a concept has both an existential and universal restriction acting on the same relation, while the relation fillers are disjoint with or complement of each other.
(d) a concept has an existential restriction acting on a relation, while the relation filler is an intersection of concepts which are disjoint with or complement of each other.
(e) a concept has an existential restriction acting on a relation, while the domain of the relation is disjoint with or complement of the concept.
(f) a concept has an existential restriction acting on a relation, while the range of the relation is disjoint with or complement of the relation filler.
In the next subsection, we illustrate how paths are used to check the contradiction in example (c) and (d). Other examples are checked analogously.

3.1 Disjoint Contradiction

Given an ontology O, a concept C and a disjoint node N, a disjoint contradiction may occur if the concept C has relationships with other concepts which are linked by the disjoint node N; informally, two of concept paths starting with C must link to disjoint paths from node N.

We first find a set of concept paths $\Phi = \{\Pi_1, \Pi_2, \ldots, \Pi_n\}$ by applying the function *concept-path*(C, Φ, O). A set of

Table 1: Six contradiction examples



disjoint paths $\Phi_d = \{\Pi_{d1}, \Pi_{d2}, \ldots, \Pi_{dn}\}$ is found by the function *disjoint-path*(N, Φ_d, O). Each concept path Π_i is compared to each disjoint path: if there exist two concept paths Π_i and Π_j, such that $i \neq j$, and there exist two disjoint paths Π_{dk} and Π_{dl} such that $k \neq l$, and *matched-tail*(Π_i, Π_{dk}) and *matched-tail*(Π_j, Π_{dl}) are both true (Definition 7), that means the concept paths Π_i and Π_j may be involved in the contradiction.

For example, Figure 6 shows are a number of nodes $\mathbf{N}$ and edges $\mathbf{E}$. Lion has an eats relation with $\exists$ restriction, and the relation filler is the intersection of Cow and Sheep; Cattle and Sheep are defined to be disjoint; Cow is a subconcept of Cattle. The set of concept paths from C is $\Phi = \{\Pi_1, \Pi_2, \Pi_3\}$, where

$$\Pi_1 = \langle C, R_1, C_1 \rangle$$
$$\Pi_2 = \langle C, R_2, N_1, C_1 \rangle$$
$$\Pi_3 = \langle C, R_2, N_1, C_2 \rangle$$

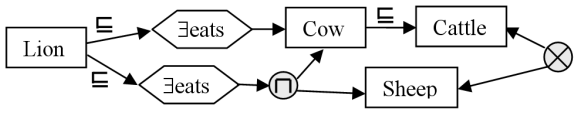
The set of disjoint paths derived from the disjoint node N is $\Phi_d = \{\Pi_{d1}, \Pi_{d2}\}$, where

$$\Pi_{d1} = \langle C_2 \rangle$$
$$\Pi_{d2} = \langle C_3 \rangle$$

As Cow is a subconcept of Cattle, the concept node C_1 in Π_1 and Π_2 must be substituted by C_3, therefore,

$$\textit{substitute}(\Pi_1, C_3, \Pi_1'), \Pi_1' = \langle C, R_1, C_3 \rangle$$
$$\textit{substitute}(\Pi_2, C_3, \Pi_2'), \Pi_2' = \langle C, R_2, N_1, C_3 \rangle$$

By comparing the concept and disjoint paths, we found that Π_1' and Π_2' are matched with Π_{d1}, as their common node is C_3; Π_3 is matched with Π_{d2}, as their common node is C_2. For the matched concept paths, their *matched-tail* and *common-subpath* are obtained. Figure 7 illustrates the *common-subpath* and *matched-tail* parts of Π_2' and Π_3.



$\mathbf{N} = \{C = \langle \mathsf{Lion}, \emptyset, \emptyset \rangle,$
$C_1 = \langle \mathsf{Cow}, \emptyset, \emptyset \rangle,$
$C_2 = \langle \mathsf{Sheep}, \emptyset, \emptyset \rangle,$
$C_3 = \langle \mathsf{Cattle}, \emptyset, \emptyset \rangle,$
$R_1 = \langle \mathsf{eats}, C, C_1, \exists \rangle,$
$R_2 = \langle \mathsf{eats}, C, N_1, \exists \rangle,$
$N_1 = \langle n_1, \sqcap \rangle,\ N_2 = \langle n_1, \otimes \rangle\ \}$
$\mathbf{E} = \{\langle C_1, \sqsubseteq, C_3 \rangle, \langle N_2, \emptyset, C_3 \rangle, \langle N_2, \emptyset, C_1 \rangle, \cdots\}$

Figure 6: lion example with a disjoint axiom

$$\textit{matched-tail}(\Pi_1', \Pi_{d2}), \Pi_{d2} = \langle C_3 \rangle$$
$$\textit{matched-tail}(\Pi_2', \Pi_{d2}), \Pi_{d2} = \langle C_3 \rangle$$
$$\textit{matched-tail}(\Pi_3, \Pi_{d1}), \Pi_{d1} = \langle C_2 \rangle$$
$$\textit{common-subpath}(\Pi_1', \Pi_2') = \langle C \rangle$$
$$\textit{common-subpath}(\Pi_1', \Pi_3) = \langle C \rangle$$
$$\textit{common-subpath}(\Pi_2', \Pi_3) = \langle C, R_2, N_1 \rangle$$

3.1.1 Detecting disjoint contradictory concept paths

Though the concept paths Π_1', Π_2', Π_3 are matched with the disjoint paths, not every matched path is relevant to the contradiction. If two concept paths Π_i, Π_j are matched with disjoint paths Π_{dl}, Π_{dk} based on the conditions mentioned in Section 3.1, i.e. *matched-tail*(Π_i, Π_{dk}) and *matched-tail*(Π_j, Π_{dl}) are both true; they are disjoint contradictory, if the following four conditions hold:

1. No $\sqcup$ node exists in the paths Π_i, Π_j, excluding their *matched-tail* parts, which are not considered because the *matched-tail* parts are the same as the disjoint paths. If a $\sqcup$ node exists, that means the path may be alternative.
2. All relation nodes in Π_i must have the same name with the corresponding nodes in Π_j, excluding their *matched-tail* parts for the same reason as (1) above. If the relation nodes in Π_i have different names with those in Π_j, that means they participate in different relationships, hence they may not have a contradiction.
3. If there exist relation nodes in *common-subpath*(Π_i, Π_j), the restriction of the relation nodes must be $\exists$. If the restriction of the relation node is $\forall$, that means the concept may contain individuals that have this relationship without linking to any other individuals.
4. Excluding the *matched-tail* and *common-subpath* parts, the relation nodes of path Π_i must have the same restriction as each other (that is, the relation nodes of Π_i must be either all $\exists$ or all $\forall$ restrictions); likewise, the relation nodes of path Π_j must have the same restriction as each other; finally, the restriction used in Π_i must be different to the one used in Π_j (for example, Π_i could use all $\exists$, while Π_j could use all $\forall$). If the relation nodes of the concept path Π_i contain both $\exists$ and $\forall$ restrictions, or if the two concept paths have the exactly same restriction of relation nodes, that means the concept may contain individuals which have the relationships without linking to any other individuals.

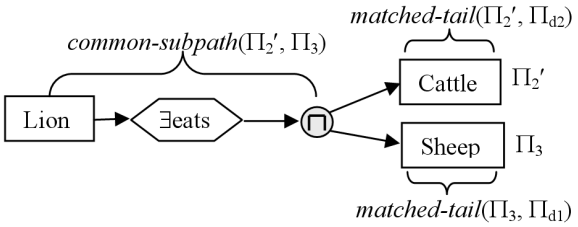


Figure 7: The matched-tail and common-subpath part of two paths

Therefore, Π_1' and Π_3 are not contradictory, as the restriction of the relation node eats in Π_1' is the same as the relation node eats of Π_3 (condition 4). Π_2' and Π_1' are not contradictory for the same reason. Π_2' and Π_3 are contradictory, *common-subpath*(Π_2', Π_3) = $\langle C, R_2, N_1 \rangle$. There is no $\sqcup$ node in the paths excluding the *matched-tail* (condition 1); both paths have the same relation nodes (condition 2); the relation eats in the *common-subpath* is $\exists$ restriction(condition 3).

3.1.2 Options to resolve the disjoint contradiction

After getting two contradictory paths, a number of options are provided to resolve the contradiction:

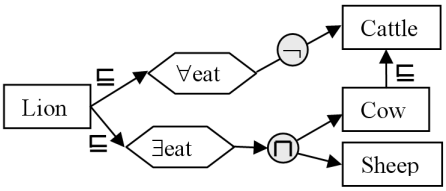
1. Modify the *common-subpath* of the contradictory paths
   - if there exists a relation node with $\exists$ restriction in the *common-subpath*, change its restriction to $\forall$. Referring to the example, if the restriction of the relation node in *common-subpath*(Π_2', Π_3) is changed to $\forall$, then $R_2' = \langle \mathsf{eats}, C, N_1, \forall \rangle$.
   - if there exists a $\sqcap$ node in the paths excluding the *matched-tail*, change it to a $\sqcup$ node. If the $\sqcap$ node in *common-subpath*(Π_2', Π_3) is changed to a $\sqcup$ node, then $R_2' = \langle \mathsf{eats}, C, N_1', \forall \rangle$, where $N_1' = \langle n_1, \sqcup \rangle$.
2. Modify the *matched-tail* part to make it different from the disjoint paths. In this example, the matched tail parts of Π_2' and Π_3 are C_3 and C_2 respectively, the concept nodes in Π_2' and Π_3 can be changed to other concepts which are not C_3, C_2, or their subconcepts.

3. Remove the disjoint node $\otimes$.

4. Remove the relationships of the concept. If R_1 is removed, the path Π_1 is removed; if R_2 is removed, then paths Π_2 and Π_3 are removed.

3.2 Complement Contradiction

This kind of contradiction occurs when a concept is defined to have a relationship with some concept and must not have a relationship with that concept, simultaneously. Given an ontology O, a concept C, we can find a set of concept paths $\Phi = \{\Pi_1, \Pi_2, \cdots, \Pi_n\}$ starting from a concept C using the function *concept-path*(C, Φ, O), in which every concept path Π_i with a $\neg$ node is compared with every concept path Π_j without a $\neg$ node. By comparing the two concept paths, and converting to negation normal form [3], if necessary, we infer which relationships of the concepts result in the contradiction based on the conditions described below.



$\mathbf{N} = \{C = \langle \mathsf{Lion}, \emptyset, \emptyset \rangle,$
$C_1 = \langle \mathsf{Cow}, \emptyset, \emptyset \rangle,$
$C_2 = \langle \mathsf{Sheep}, \emptyset, \emptyset \rangle,$
$C_3 = \langle \mathsf{Cattle}, \emptyset, \emptyset \rangle,$
$R_1 = \langle \mathsf{eat}, C, N_1, \forall \rangle,$
$R_2 = \langle \mathsf{eat}, C, N_2, \exists \rangle,$
$N_1 = \langle n_1, \neg \rangle,\ N_2 = \langle n_2,\ \sqcap \rangle\}$
$\mathbf{E} = \{\langle C_1, \sqsubseteq, C_3 \rangle, \langle C, \sqsubseteq, R_1 \rangle, \langle C, \sqsubseteq, R_2 \rangle, \cdots\}$

Figure 8: lion example with a complement axiom

For example in Figure 8, Lion is defined to have at least one eat relationship with an individual which is both a member of Cow and Sheep, and it is also defined that its individuals only have the eat relationship to other individuals which are not members of Cattle. Also Cow is a subconcept of Cattle. These statements are contradictory. The set of concept paths are $\Phi = \{\Pi_1, \Pi_2, \Pi_3\}$, where
$\Pi_1 = \langle C, R_1, N_1, C_3 \rangle$
$\Pi_2 = \langle C, R_2, N_2, C_1 \rangle$
$\Pi_3 = \langle C, R_2, N_2, C_2 \rangle$

As Cow is a subconcept of Cattle, the concept node C_1 in Π_2 must be substituted by C_3, therefore, *substitute*(Π_2, C_3, Π_2'), $\Pi_2' = \langle C, R_2, C_3 \rangle$. For the convenience of comparison, R_1 is converted to negation normal form, i.e. $R_1' = \langle \mathsf{eat}, N_1, C_3, \exists \rangle$, $\Pi_1' = \langle C, N_1, R_1', C_3 \rangle$ (see Figure 9).
It is obvious that Π_1' and Π_2' are contradictory, as Lion is defined not to have individuals which have at least

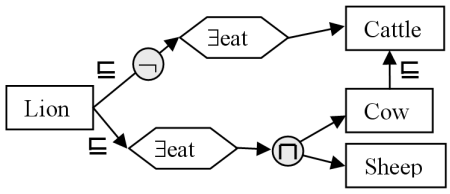


Figure 9: lion example converted to negated normal form

one eat relationship to an individual which is a member of the concept Cattle; Lion is also defined to have individuals that have at least one eat relationship with an individual that is a member of the concept Cattle.

For the convenience of comparison, the relation nodes in a path, which have different restrictions with another path, are converted to the same restriction by negation normal form. The two concepts paths Π_i and Π_j are complement contradictory if the following four conditions hold:

1. No $\sqcup$ node exists in the paths. If a $\sqcup$ node exists, then the paths may be alternative.

2. All relation nodes in path Π_i must have the same name as the corresponding nodes in Π_j. If they have different relation names, this means the individuals of the concept in the paths participate in different relationships.

3. The relation nodes which are preceding the $\neg$ node in path Π_i must have the same name but different restrictions with the corresponding nodes in Π_j. If those relation nodes preceding the $\neg$ node in the two paths have the same restriction, this means the concept may contain the individuals that have relationships without linking to any other individuals.

4. The relation nodes which exist after the $\neg$ node in path Π_i must have the same name and restriction with the corresponding nodes in Π_j. If the relation nodes after the $\neg$ node have different restrictions, this means the individuals of the concept in the two paths can participate in different relationships.

After getting two contradictory paths, a number of options are provided to resolve the contradiction:

1. If there exists a $\sqcap$ node in the paths, change it to be $\sqcup$ node, the path then becomes alternative. In the above example, if the $\sqcap$ node is changed to be $\sqcup$, then $R_2 = \langle \mathsf{eat}, C, N_2', \exists \rangle$, where $N_2' = \langle n_2, \sqcup \rangle$.

2. Change the restriction of one of the relation nodes in either one of the paths. If the restriction of

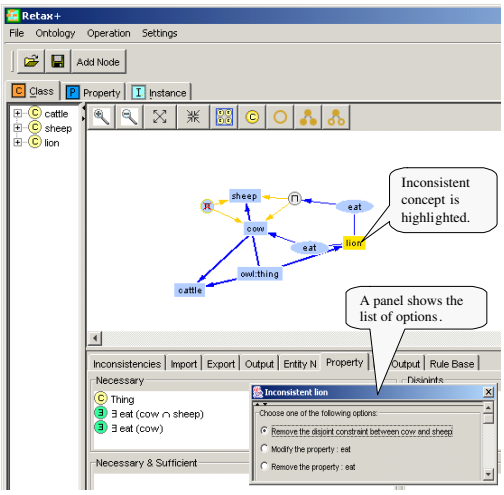


Figure 10: Screen Shot of ReTAX++

the eat relation is changed to be $\forall$, then $R'_2 = \langle \mathsf{eat}, C, N_2, \forall \rangle$. Therefore, the concept Lion may contain individuals that do not have any eat relationship to any other individuals.

3. Remove the complement axiom by eliminating the $\neg$ node in a path. If the $\neg$ node in path Π_1 is removed, then $R'_1 = \langle \mathsf{eat}, C, C_3, \forall \rangle$

4. Remove one of the relationships of the concept C. In this case, either R_1 or R_2 can be removed.

4. IMPLEMENTATION

A screenshot of the ReTAX++ is shown in Figure 10. The ontology is displayed in a graphical format using TouchGraph[1]. By interacting with an external reasoner, Racer[2], the system detects and highlights inconsistent concepts. When the user clicks the inconsistent item, a list of options to resolve the problem is shown. The changes in the ontology are implemented automatically. Though a number of options to resolve inconsistencies are provided, we are not sure how useful the suggestions are to the ontologists. An empirical study will be conducted to evaluate the usability and efficiency of the system, so that the facilities of resolving inconsistencies can be further improved.

5. RELATED WORK

The evaluation and maintenance of ontologies has been discussed extensively in the literature. The approach of minimising common errors in ontology development is presented by Rector et al. [1] in the CO-ODE project. They then presented a heuristic approach for debugging OWL ontologies called OWLDebugger, which alleviates the user from troubleshooting the inconsistent classes. It helps users to track down the reasons for inconsistencies in OWL classes [7], descriptions that explain the reasons for inconsistent OWL classes are also generated [5]. However, it cannot determine the root causes of unsatisfiability in every case, as they use heuristic approaches and pattern matching. Further, the system lists all unsatisfiable concepts without identifying the root and derived unsatisfiable concepts, the user has to debug one by one and run the reasoner frequently to check the consistency. In the contrast, the graph-based approach shows the relationships among concepts graphically, users finds it easy to understand the problem even without reading the natural language explanations. The suggestions of solving inconsistency further alleviates the user from debugging the ontology.

Schlobach [11] proposed a strategy for automatically identifying and fixing the incoherence by pinpointing. In pinpointing they identify minimal sets of axioms which need to be removed or ignored to make an ontology coherent. In the case of numerous inconsistent concepts, it chooses (and eliminates) axioms that most frequently participate in the underlying logical contradictions. Their case studies showed this strategy is useful for dealing with the incoherence due to the merging of two or more ontologies.

Haase et al. [6] introduced two resolution strategies to ensure that consistency is maintained as the ontology evolves. The first strategy is to start out with an inconsistent ontology and iteratively remove axioms until a consistent ontology is obtained. The second approach is to find a minimal inconsistent ontology and present it to the user. The user can then decide how to resolve the inconsistency. Its main feature is to preserve consistency in the case that consistency conditions are violated in the presence of ontology changes.

Schlobach and Haase proposed mechanisms to detect inconsistencies and remove the axioms which are the root causes of inconsistencies. Though they can get a consistent ontology, some information may be lost due to removing or ignoring axioms. The second approach by Haase is to present the inconsistencies to the user, but it still cannot alleviate the user from troubleshooting the problem. To compensate for this, our strategies can point out which relationships of the inconsistent concepts lead to contradictions, and help the user to correct the content, where the relation restrictions, the conjunction axioms and complement axioms etc., can be modified or removed.

6. CONCLUSION AND DISCUSSION

This paper highlights a formalism used to represent ontologies graphically. The system detects the inconsistent concepts by interacting with a reasoner, and provides options to resolve the problems. By adopting a graph-based approach to representing ontologies, we can identify the relationships of the concepts which result in inconsistencies by analysing paths of the graph.

[1]http://www.touchgraph.com/
[2]http://www.racer-systems.com

The user is provided with facilities to rectify the relationships or axioms of concepts to eliminate the incoherence.

There are other points for future investigations. We believe our algorithms work efficiently in dealing with ontologies with a small number of inconsistent concepts. However, an ontology may contain a large number of inconsistent concepts due to inconsistencies propagated from another source. For example, the unsatisfiability of a concept C is propagated to all of its subconcepts. It is not practical to check inconsistent concepts one by one, this is because the unsatisfiable concept C is resolved, it is not necessary to debug all of its subconcepts. Therefore, a strategy to discover the root causes of inconsistent concepts is important to increase the efficiency. Besides, when a concept with numerous relationships (such as $C \sqsubseteq (C_1 \sqcup C_2) \sqcap \ldots \sqcap (C_n \sqcup C_m)$) is inconsistent, the optimization of comparing paths has to be investigated to guarantee the efficiency. Another problem is how to guarantee the consistency after changes. We will formally prove that the graph-based algorithms indeed capture semantic problems within the ontology and prove that the semantics of the ontology will hold after the corrections are made. Also, if the user ignores the suggested options and makes his own changes, it may induce other inconsistencies; therefore the propagated changes are presented to the user. To avoid performing undesired changes, before the system applies a change to the ontology, it will generate and present the user with a list of all the implications of the proposed change(s).

7. REFERENCES

[1] M. Rogers J. Knublauch H. Stevens R. Wang A. L. Rector, N. Horridge and C. Wroe. Designing user interfaces to minimise common errors in ontology development: The co-ode and hyontuse projects. In *UK E-Science All Hands Meeting 2004(AHM04)*. ACM Press, August 2004.

[2] K. Baclawski, C. J. Matheus, M. M. Kokar, J. Letkowski, and P. A. Kogut. Towards a symptom ontology for semantic web applications. In *The Semantic Web - ISWC 2004: Third International Semantic Web Conference*, Lecture Notes in Computer Science, pages 650–667. Springer, 2004.

[3] F. Badder, D. Calvanese, D. L. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[4] Ó. Corcho, A. Gómez-Pérez, R. González-Cabero, and M. C. Suárez-Figueroa. Odeval: A tool for evaluating rdf(s), daml+oil, and owl concept taxonomies. In *First IFIP Conference on Artificial Intelligence Applications and Innovations*. Toulouse, France, 2004.

[5] A. Rector N. Drummond H. Wang, M. Horridge and J. Seidenberg. A heuristic approach to explain the inconsistency in owl ontologies. In *8th International Protégé Conference*, 2005.

[6] Peter Haase and Ljiljana Stojanovic. Consistent evolution of owl ontologies. In *ESWC*, pages 182–197, 2005.

[7] M. Horridge. A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0. Technical report, University of Manchester, 2004.

[8] SC. Lam, D. Sleeman, and W. Vasconcelos. Retax+: a cooperative taxonomy revision tool. In *The Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 64 – 77. Springer, 2004.

[9] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, 2000.

[10] Stefan Schlobach. Debugging and semantic clarification by pinpointing. In *ESWC*, pages 226–240, 2005.

[11] F. van Harmelen Z. Huang and A. ten Teije. Reasoning with inconsistent ontologies. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, Edinburgh, Scotland, August 2005.