UNIVERSITY OF INFORMATION AND TECHNOLOGY

Báo cáo khóa luận giai đoạn 8-10/2014

hoàn thành bởi

với sự hướng dẫn của

TS. Nguyễn Anh Tuấn

Đề tài khóa luận
tốt nghiệp - sinh viên khóa MMTT2010

thuộc
Chuyên ngành phát triển ứng dụng Web và di động
Khoa mạng máy tính và truyền thông

Ngày 25 tháng 10 năm 2014

Mục lục

Abbreviations

KB	**K**nowledge **B**ase
KR	**K**nowledge **R**epresentation
DL	**D**escription **L**ogic
SHOIN	

Symbols

$\models$	models of	có nghĩa trong (KB)
$\subseteq$	is a subset of	là tập con của
$\cap$	intersect	giao với
$\neg\ A$	complement of A of	không phải A
$\exists\ R.E$	e.g *has* some E	
$\forall\ R.E$	e.g *has* only E	

For/Dedicated to/To my...

Chương 1

Các nguyên nhân dẫn đến tính thiếu nhất quán trong ontology

1.1 Các định nghĩa cần lưu ý[3]

Unsatisfiable Class/Concept dùng để chỉ một lớp hay một khái niệm trong một ontology mà ngữ nghĩa xung đột với ngữ nghĩa khác được nêu ra trong ontology hay có thể nói là các phát biểu về lớp hay khái niệm này mâu thuẫn với nhau hoặc mâu thuẫn với những phát biểu khác trong ontology.

Ví dụ `Cow`

```
        SubClassOf: Vegetarian
Vegetarian
    SubClassOf: Animal and eats only Plant
DisjointClasses:
    Plant, Animal
```

Giải thích Trong ví dụ trên thì MadCow chính là một lớp không hợp lý do trong các phát biểu logic của nó mâu thuẫn với nhau Cow là lớp con của Vegeterian mà Vegeterian chỉ ăn Plant (nghĩa là ngoài Plant, Vegeterian không ăn thứ gi khác) trong khi đó khai báo của lớp MadCow là lớp con của Cow và ăn một số Sheep (Sheep là một lớp con của Animal).

Từ đó việc lý luận có thể đưa ra giả định sai là Sheep cũng có khả năng là một phần của Plant . Điểm quan trọng là Plant và Animal là 2 DisjointClasses, nói cách khác không tồn tại một cá thể nào vừa thuộc lớp Plant và vừa thuộc lớp Animal. Như vậy trong tất cả các phát biểu logic ở ví dụ trên đã có 2 phát biểu

gây mâu thuẫn chính là eats only Plant và eats some Sheep, và chúng làm cho lớp MadCow trở nên bất hợp lý (unsatisfiable).

Incoherent Ontology dùng để chỉ một *ontology/model* có ý nghĩa không mạch lạc rõ ràng do nó có chứa ít nhất một *Unsatisfiable Class/Concept* và với điều kiện là trong những *Unsatisfiable Class* này không được chứa bất kì một cá thể (*Individual*) nào.
Giả sử ta có ontology A chứa các các phát biểu trong ví dụ trên ngoại trừ phát biểu cuối cùng `Individual: Dora type: MadCow` thì ta có thể nói ontology A không mạch lạc rõ ràng do nó chứa unsatisfiable class là MadCow. Chúng ta vẫn có sử dụng được ontology A vì nó vẫn còn tính nhất quán (*Consistency*)miễn là không có phần tử nào thuộc lớp MadCow.

Inconsistent Ontology dùng để chỉ một ontology chứa ít nhất một *Unsatisfiable Class* và có ít nhất một cá thể(*Individual*) thuộc một trong những lớp *unsatisfiable* này. Như đã thể hiện trong ví dụ đầu tiên thì cá thể Dora thuộc lớp MadCow (Một lớp unsatisfiable thì không nên phép có bất kì cá thể nào nếu như chúng ta muốn đảm bảo tính consistency cho ontology), như vậy bất kì ontology nào có những phát biểu trên đều được coi là không nhất quán (*inconsistency*), đều này đồng nghĩa là ontology đó không thể sử dụng được nữa.

1.2 Các nguyên nhân phổ biến dẫn đến tính thiếu nhất quán (Inconsistency)[4]

Các nguyên nhân dẫn đến tính thiếu nhất quán trong ontology gây bởi các lỗi được phân loại thành lỗi gây ra bởi phát biểu ở mức độ lớp (Class level - TBox), các lỗi gây ra bởi phát biểu ở mức độ cá thể (Instance/Individual level - ABox) và lỗi gây ra bởi sự kết hợp của cả 2 nguyên nhân vừa nêu trên.

1.2.1 Khởi tạo cá thể cho một Unsatisfiable Class) - (TBox + ABox)

- Khởi tạo cá thể cho một Unsatisfiable Class được xem là nguyên nhân phổ biến nhất gây ra tính thiếu nhất quán trong ontology.
- Ví dụ:

```
Individual: Dora type: MadCow
```

- Chúng ta không quan tâm đâu là nguyên nhân làm cho `MadCow` trở nên mâu thuẫn, chỉ cần biết là một Unsatisfiable Class thì không nên có bất kì cá thể nào trong đó. Rõ ràng là không có bất kì ontology nào mà cá thể Dora có thể đáp ứng các điều kiện như trong ví dụ đầu tiên, nói cách khác không tồn tại model nào có thể thỏa được điều kiện trên. Chúng ta phát biểu đó là một ontology không nhất quán.

1.2.2 Khởi tạo cá thể thuộc 2 class được disjoint với nhau (TBox + ABox)

- Đây là một trường hợp dễ bắt gặp vì nó sai ngay trong phát biểu về logic.
- Ví dụ

```
Individual: Dora
Types: Vegetarian, Carnivore
DisjointClasses: Vegetarian, Carnivore
```

- Lớp A disjoint với B khi và chỉ chi lớp A không có chung bất kì một phần tử/cá thể nào với lớp B. Phát biểu Disjoint Classes(A B C) có nghĩa là mỗi lớp trong đó disjoint với từng lớp còn lại (mutually disjoint). Phát biểu ABox dạng `DisjointClasses(Vegetarian Carnivore)` là sai vì Dora vừa thuộc Vegetarian vừa thuộc Carnivore dựa vào phát biểu `Individual: Dora Types: Vegetarian, Carnivore`.

1.2.3 Các phát biểu ABox xung đột với nhau

- Trường hợp này thì tương tự như nguyên nhân ở trên nhưng khác ở chỗ là lần này sự mâu thuẫn nằm trong các biểu ở cấp độ cá thể (ABox).
- Ví dụ:

```
Individual: Dora
   Types: Vegetarian, not Vegetarian
```

- Dễ thấy được sự mâu thuẫn trong trong phát biểu trên vừa yêu cầu Dora là Vegetarian vừa yêu cầu nó không phải Vegetarian.

1.2.4 Phát biểu xung đột với nghĩa "oneOf" (All TBox)

- Phát biểu bao gồm hoặc một trong (oneOf trong cú pháp của OWL) cho phép sử dụng các cá thể trong khai báo phát biểu ABox, sự kết hợp này có thể dẫn đến sự thiếu nhất quán.

- Lấy ví dụ sau:

```
Class: MyFavouriteCow
   EquivalentTo: {Dora}
Class: AllMyCows
   EquivalentTo: {Dora, Daisy, Patty}
DisjointClasses: MyFavouriteCow, AllMyCows
```

- Phần đầu tiên của các phát biểu trên tất cả các thể thuộc lớp `MyFavouriteCow` phải tương đương với cá thể tên Dora, nói cách khác là `SameIndividual` với `Dora`. Phần thứ hai cũng tương tự tất cả các cá thể thuộc lớp `AllMyCows` buộc phải tương đương với một trong 3 cá thể tên `Dora`, `Daisy` hoặc `Patty`. Do 2 phát biểu trên chúng ta đã nói Dora thuộc cả 2 lớp `MyFavoriteCow` và `AllMyCows` nên mâu thuẫn với phát biểu cuối cùng khi nói 2 lớp này không có chung một cá thể nào. Vì vậy dẫn tới ontology bị thiếu nhất quán (inconsistent).

1.2.5 Không có khả năng khởi tạo bất kì cá thể nào (all TBox)

- Ví dụ:

```
Vegetarian or not Vegetarian
SubClassOf: Cow and not Cow
```

- Đây chỉ là một ví dụ đơn giản để minh họa cho trường hợp này. Thực tế sẽ ít người dùng nào tạo ra một phát biểu ngớ ngẩn như vậy nhưng nó vẫn có khả năng xảy ra khi phát biểu trên là kết quả từ suy luận (reasoning) của những phát biểu lớn và phức tạp hơn.

- Có thể giải thích ví dụ trên như sau. Đầu tiên để đáp ứng ý nghĩa dòng đầu tiên yêu cầu cá thể vừa là `Vegetarian` hoặc không phải `Vegetarian` - bất kỳ phát biểu nào dạng này, "cá thể thuộc hoặc không thuộc một lớp" chính là tất cả cá thể xuất hiện trong ontology. Dòng thứ hai yêu cầu cá thể vừa là Cow vừa không phải là Cow, phát biểu này rơi vào một trong các nguyên nhân vừa nêu ở trên. Tổng hợp lại chúng yêu cầu tất cả cá thể vừa là Cow vừa không phải Cow, điều này gây ra mâu thuẫn trên toàn ontology do phát biểu đầu tiên chỉ tới tất cả các cá thể.

Kết luận Trên đây chúng em đã liệt kê những nguyên nhân phổ biến dẫn đến thiếu nhất quán qua những ví dụ đã được đơn giản hoá để dễ dàng nắm bắt được đâu là căn nguyên gây ra sự mâu thuẫn về logic. Trên thực tế với những ontology có số lượng phát biểu lớn và phức tạp rất khó để người dùng có thể nhận diện được đâu là nguyên nhân

chính xác gây ra mâu thuẫn, do vậy sự ra đời của một công cụ giúp chúng ta phát hiện chính xác nguyên nhân gây lỗi là rất cần thiết. Vì vậy trong nội dung chương 2, chúng em sẽ đề cập tới Ontology Debugging một khía cạnh rất được chú trọng khi số lượng phát biểu của ontology ngày càng tăng.

Chương 2

Giải pháp để sửa chữa inconsistent ontology

- Như đã được đề cập trong chương 1, trong các nguyên nhân dẫn đến tính thiếu nhất quán (*Inconsistency*) trong ontology thì **Unsatisfiable Class** (lớp không thỏa về tính logic) là nguyên nhân nếu có thể được phát hiện sớm để loại bỏ hoặc sửa lại các phát biểu gây mâu thuẫn thì giúp cho ontology tránh bị inconsistent.

- Đã có rất nhiều nghiên cứu thành công trong việc tìm và phát hiện lỗi (các phát biểu mâu thuẫn) trong ontology. Trong đó có một nghiên cứu nổi bật[1], không chỉ có khả năng phát hiện gần như chính xác các nguyên nhân gây lỗi mà còn được đưa ra các giải pháp tối ưu* để sửa lỗi. Nghiên cứu này đã được ứng dụng để đưa ra các giải thích về các lớp không thỏa về nghĩa (unsatisfiable classes) trong bộ thự viện lập trình ontology thông dụng hiện nay là OWL-API[2]. Sau đây chúng em xin được trình bày lại những điểm quan trọng trong nghiên cứu vừa được đề cập**

* *Tối ưu có nghĩa là hạn chế tối đa các thay đổi về ý nghĩa mà việc xóa hoặc thay đổi phát biểu mâu thuẫn có thể gây ra cho các phát biểu khác (other axioms) trong ontology.*

** *Mọi quan điểm và ý tưởng trình bày ở phần sau của Chương 2 đều thuộc sở hữu của các tác giả bài báo[1] và [9]. Chúng em chỉ trình bày lại sau khi đã đọc và nắm được ý tưởng chính yếu của bài báo.*

2.1 Mục tiêu của việc debuging ontology

Mục tiêu chính của việc debuging ontology gồm hai phần quan trọng. Thứ nhất, với một ontology có số lượng lớn các lớp unsatisfiable, cần tìm và nhận dạng được nguyên nhân gây ra mâu thuẫn và các lớp bị ảnh hưởng bởi sự mâu thuẫn đó trong ontology. Thứ hai, cho biết trước một Unsatisfiable Class cụ thể, trích xuất và trình bày cho người sử dụng ontology(*modeler*) một tập hợp tối tiểu các phát biểu (*minimal set of axioms*) từ ontology hay nguyên nhân chính xác chịu trách nghiệm trong việc gây ra sự mâu thuẫn về logic.

2.2 Khái niệm và các kỹ thuật cần biết

Các hệ thống Description Logic thường cung cấp một tập hợp các tác vụ suy luận đã được chuẩn hóa như phân loại các khái niệm (*concept classification*), kiểm tra tính đáp ứng về logic (*concept satisfiability*) và kiểm tra tính nhất quán của knowledge base (KB). Hầu hết các reasoner thông dụng hiện nay đều buộc phải cung cấp đủ 3 tác vụ nêu trên, nhưng tất cả chúng đều không thân thiện với người dùng. Do tất cả những gì chúng ta biết được đều là kết quả (hay output) từ sự suy luận(reasoning) của reasoner.

Để giúp cho các tác vụ suy luận (reasoning) trở nên thân thiện với người dùng hơn, một hệ thống DL-based Knowledge Representation (KR) phải mở rộng thêm các lựa chọn về các tác vụ không nằm trong tiêu chuẩn của DL. Một ví dụ cụ thể là việc tạo ra các giải thích tại sao một lớp lại bị reasoner đánh giá là unsatisfiable. Thêm một tình huống mà người dùng cần được giải thích là tại sao reasoner đánh giá một lớp là lớp con của một lớp khác - đâu là lý do. Việc ra đời tác vụ giải thích nguyên nhân và kết quả là thật sự cần thiết trong bối cảnh sự phát triển nhanh của Semantic Web và cộng đồng người dùng/nhà phát triển Ontology ngày càng tăng nhanh.

2.2.1 Dịch vụ Axiom Pinpointing

Axiom Pinpointing service chính là dịch vụ có khả năng thực hiện tác vụ giải thích vừa được đề cập, với một KB và bất kì kết quả suy luận nào từ KB, dịch vụ này sẽ trả về tập các chứng minh/giải thích cho suy luận đó bằng những phát biểu đã được khai báo trong KB.

Có thể giải thích ngắn gọn như sau, cho một phát biểu kết quả họ SHOIN α được suy ra từ một knowledge base K, tập các giải thích/chứng minh cho α trong K là một phần tối tiểu $K' \subseteq K$ chịu trách nhiệm cho α xảy ra. Chứng minh K' là tối tiểu với

điều kiện α là một kết quả logic được suy ra từ $K^{'}$, hay nói cách khác $K^{'}$ tối tiểu khi và chỉ khi bất kì tập con nào của $K^{'}$ đều không suy ra được α. Nói chung có thể tồn tại nhiều giải thích/chứng minh cho α trong K.

Sau đây là một ví dụ cho ý tưởng vừa nêu. Cho KB K với các phát biểu như sau:

1. $A \subseteq B \cap C$
2. $B \subseteq \neg\ E$
3. $A \subseteq D \cap \exists\ R.E$
4. $D \subseteq C \cap \forall\ R.B$

Trong KB trên, A, B, C, D, E là atomic concepts và R là atomic role. Chúng ta sẽ dùng số thứ tự của từng câu phát biểu trên thay vì lặp lại nguyên văn.

Từ các phát biểu trên ta có $K \models (A \subseteq C)$. Tuy nhiên, điều kiện cần và đủ để suy ra được một kết quả tương tự từ 2 phần nhỏ hơn của KB K là $K_1 = 1$ và $K_2 =$3,4. Chúng ta nói K_1 và K_2 là các giải thích/chứng minh cho kết luận nói C là tập con của A - $A \subseteq C$.

KB trong ví dụ vừa nêu được xem là khá nhỏ, qua đó dễ dàng nhận ra lợi ích đáng kể khi số lượng phát biểu trong KB tăng lên vài trăm hay vài ngàn phát biểu. Bằng cách nhận dạng chính xác các tập tối tiểu chứa các phát biểu khẳng định (asserted) là những giả thiết cho kết quả được suy ra, dịch vụ này có thể được dùng để cô lập, đánh dấu và giải thích nguyên nhân hoặc cơ sở của các kết quả suy luận. Điều này cực kì quang trọng trên khía cạnh debugging, lấy ví dụ trường hợp cần giải thích là một Unsatisfiable Class/Concept, dịch vụ này sẽ khám phá tất cả và chỉ những phát biểu là nguyên nhân gây lỗi. Trong trường hợp vừa nêu, để sửa lại unsatisfiable class cần loại bỏ ít nhất một phát biểu trong tập các phát biểu tối tiểu nguyên nhân gây lỗi MUPS - sẽ được đề cập trong mục bên dưới.

Tuy nhiên, dịch vụ axiom pinpointing chúng ta đề cập có một giới hạn là nó chỉ làm việc ở mức độ các phát biểu với nhau chứ chưa phân biệt được phần cụ thể nào của phát biểu mới là nguyên nhân hay cơ sở cho kết quả suy luận. Lấy lại ví dụ vừa nếu trên KB K, lớp B trong giao của $B \cap C$ trong phát biểu 1, không phải là giải thiết cần để suy ra $A \subseteq C$. Tương tự, $\exists\ R.E$ và $\forall\ R.B$ trong phát biểu 3 và 4 cũng không liên quan hay giúp suy ra được $A \subseteq C$. Việc quan tâm xem phần nào của phát biểu mới chính là giả thiết/nguyên nhân của kết quả suy luận rất quan trọng trong nhiều trường hợp, đặc biệt khi sửa chữa một phát biểu gây lỗi liên quan tới việc sửa lại phát biểu hơn là xóa nó đi.

Để đáp ứng yêu cầu này, họ để định nghĩa một *hàm chia nhỏ KB*. Ý tưởng là viết lại phát biểu trong KB thành những dạng đơn giản và dễ hiểu hơn. Sau đó áp dụng

dịch vụ Axiom Pinpointing lên những phần KB K_s đã được chia nhỏ nhất có thể, với ý nghĩa tương đương với KB K.
Ví dụ: $A \subseteq C \cap D$ tương đương với $A \subseteq C$, $A \subseteq D$
Trong một vài trường hợp, kĩ thuật này bắt buộc phải giới thiệu ra một tên lớp mới, chỉ với mục đích là chia các phát biểu thành những mảnh nhỏ hơn.
Ví dụ: $A \subseteq \exists\ R.\ (C \cap D)$ không tương đương với $A \subseteq \exists\ R.C$, $A \subseteq \exists\ R.D$
Để chia nhỏ phát biểu trên chúng ta sẽ giới thiệu một tên lớp mới, gọi là E.
Như vậy ta có: $A \subseteq \exists\ R.\ (C \cap D)$ sẽ tương đương với $A \subseteq \exists\ R.E$, $E \subseteq C$, $E \subseteq D$, $C \cap D \subseteq E$ Để thực hiện được cái gọi là *"hàm chia nhỏ KB"* họ đã đề xuất các giải thuật cho việc xác định phát biểu , cùng với đáp ứng sự chính xác và đầy đủ các các chứng minh. Các giải thuật này có thể được chia thành 2 nhóm:

1. Các giải thuật *Reasoner Dependent(or Glass-box)* được xây dựng trên quy trình đưa ra quyết định tableau dành cho Description Logic. Để đưa chúng ứng dụng với thực tế đòi hỏi phải có những chỉnh sửa triệt để và đáng kể bên trong những reasoner hiện nay.

2. Các giải thuật *Reasoner Independent(or Black-box)* chỉ sử dụng các DL reasoner cho những tác vụ con và không đòi hỏi phải chỉnh sửa lại các reasoner. Reasoner lúc này có chức năng như một "chiếc hộp đen" chấp nhận các input là class/concept và một KB và trả về output là một câu trả lời xác nhận hay phủ định, phụ thuộc vào concept này có satisfiable hoặc có nghĩa (models of) trong KB hay không. Để thu được các chứng minh/giả thiết, giải thuật dạng này lựa chọn những inputs thích hợp với DL reasoner và diễn giải từng phần của outputs sinh ra từ giải thuật. **Lưu ý:** để nắm rõ hơn xin đọc kỹ [4]

2.2.2 Minimal Unsatisfiability Preserving Sub-TBoxes (MUPS)

Khái niệm MUPS lần đầu được giới thiệu trong[6].Như đã được để cập trong phần đầu của mục này, một MUPS thật ra chính là một phần nhỏ nhất của KB K mà trong đó lý giải tại sao một lớp lại unsatisfiable, nói cách khác một MUPS là một tập tối tiểu các phát biểu mà trong đó các phát biểu này giải thích chính xác nguyên nhân gây ra mâu thuẫn về logic(unsatisfiable). Một lớp unsatisfiable có thể có nhiều MUPS trong KB K (hay cụ thể là trong ontologies). Tìm tất cả MUPS của một lớp không đáp ứng là một nhiệm vụ cực kì khó đứng ở khía cạnh debugging, từ đó để sửa lại một lớp không đáp ứng(*unsatisfiable class*) chúng ta cần loại bỏ tối thiểu ít nhất một phát biểu từ từng tập các phát biểu tối tiểu MUPS lý giải cho unsatisfiable class đó.

2.3 Các bước sửa chữa các phát biểu bị lỗi

2.3.1 Tìm tất cả các MUPS của một Unsatisfiable Class

Như vừa nói ở trên MUPS thật ra chính là một phần nhỏ nhất trong KB khiến cho một lớp unsatisfiable. Do vậy tìm và xác định MUPS chính là tìm và xác định các tập tối tiểu các phát biểu cho một lớp được suy luận là unsatisfiable. Chúng ta sẽ sử dụng *Axiom Pinpointing Service*[8] để tìm MUPS với các bước tương tự đã được mô tả chi tiết trong mục trên. Nhiệm vụ tìm kiếm *precise* MUPS của lớp không đáp ứng trong KB **K** được đơn giản hóa thành vấn đề tìm MUPS trong những phiên bản đã được tách nhỏ trong KB K_s.

2.3.2 Chiến thuật xếp hạng các phát biểu (*Axioms*)

Đây là một giai đoạn khá quan trọng trong quá trình chỉnh sửa lại các phát biểu gây lỗi, quyết định xem nên loại bỏ phát biểu nào từ MUPS để sửa được lớp được satisfiable.
Với mục tiêu này, một nhân tố đáng quan tâm là các phát biểu trong MUPS có thể được *xếp hạng* dựa theo mức độ quan trọng của chúng. Việc sửa chữa các nguyên nhân gây lỗi được trở thành một vấn đề cần được tối ưu với mục tiêu quan trọng nhất là loại bỏ tất cả các lỗi gây tính thiếu nhất quán trong ontology, trong khi vẫn chắc chắn rằng những phát biểu có thứ hạng cao, nói cách khác là có giá trị quan trọng về nghĩa sẽ được ưu tiên giữ lại và các phát biểu có thứ hạng thấp nhất sẽ bị loại bỏ.
Tiêu chí đơn giản nhất để xếp hạng các phát biểu là đếm số lần chúng xuất hiện trong MUPS từ những lớp không đáp ứng xuất hiện trong một ontology,

2.3.3 Đưa ra các nguyên nhân gây lỗi chính xác bằng giải thuật HST của Reiter

Qua các bước trên, họ đã đưa ra một quy trình để tìm được ...

Tìm MUPS bằng giải thuật HST của Reiter Một kỹ thuật tối ưu hơn kỹ thuật **Axiom Pinpointing** đã được đề cập ở mục trên nhằm tìm kiếm MUPS của một unsatisfiable class, họ đã sử dụng giải thuật Hitting Set của Reiter[7], nhằm để xác định căn nguyên(*root cause*) của một vấn đề từ một bộ(*collection*) gồm nhiều tập hợp đụng độ chứa các nguyên nhân dẫn tới vấn đề, giải thuật này sẽ tạo ra những tập tối tiểu (*minimal hitting set*) chứa các nguyên nhân gây ra vấn đề. Một tập hợp đụng độ (*hitting set*) trong một bộ **C** các tập hợp là tập hợp giao (có chung phần tử) với từng tập hợp

trong **C**. Một tập hợp đụng độ là tối tiểu nếu không có bất kì tập con nào của nó lại là một tập đụng độ cho **C**. Để áp dụng vào trường hợp này khi chúng ta đã có một bộ chứa tập hợp chứa các nguyên nhân gây lỗi(*MUPS*), họ áp dụng giải thuật này để tạo ra các tập đụng độ từ MUPS - ý tưởng ở đây là loại bỏ tất cả các phát biểu trong tập đụng độ tối tiểu sẽ giúp loại bỏ từng phát biểu gây lỗi trên từng tập phát biểu của MUPS và cuối cùng giúp cho sửa chữa được unsatisfiable class.

Chương 3

Tài liệu tham khảo

1. Aditya Kalyanpur, Bijan Parsia, Evren Sirin , Bernardo Cuenca-Grau.Repairing Unsatisfiable Concepts in OWL Ontologies, 2007. Available online at http://www.cs.ox.ac.uk/people/bernardo.cuencagrau/publications/repair.pdf

2. The OWL API. The OWL API is a Java API for creating, parsing, manipulating and serialising OWL Ontologies. Available online at https://github.com/owlcs/owlapi

3. Pellet - OWL 2 Reasoner for Java.Pellet is an OWL 2 reasoner. Pellet provides standard and cutting-edge reasoning services for OWL ontologies. Available online at http://clarkparsia.com/pellet/

4. Robert Stevens. Ontogenesis, (I can't get no) satisfiability Available online at http://ontogenesis.knowledgeblog.org/1329

5. Samantha Bail.Ontogenesis, Common reasons for ontology inconsistency.Available online at http://ontogenesis.knowledgeblog.org/1343

6. S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In Proceedings of IJCAI, 2003.

7. R. Reiter. A theory of diagnosis from first principles. 1987. Artificial Intelligence 32:57-95.

8. A. Kalyanpur, B. Parsia, B. Cuenca-Grau, and E. Sirin. Axiom pinpointing: Finding (precise) justifications for arbitrary entailments in SHOIN (owl-dl). Technical report, UMIACS, 2005-66, 2006. Technical Report