CS486/686 – Introduction to Artificial Intelligence – Fall 2016

Assignment 2: An OWL Ontology for a Question-Answering Application

Son, Dongwoo ( 20420487 )

Cho, Hyewon (

(a) Give a brief (one-paragraph is sufficient) description of your domain and question-answering application, why you chose an existing type of QA system, or why you decided to invent a new QA application.

The domain for our team's ontology is a doctor diagnosing disease based on symptoms. This ontology can be automated to answer questions as following:

- Does [ disease A ] have symptom [ symptom X ] ?

- Is [ symptom X ] a symptom of [ disease A ] ?

- Is [ disease A ] infectious ?

- Does [ disease A ] need to have a surgery ?

- What diseases have the symptom [ symptom X ] ?

- What types of diseases could the symptom [ symptom X ] possibly be ?

It can be developed further to answer questions like:

- What caused [ disease A ] to occur ?

- What are good food for [ disease A ] to cure ?

- What perscriptions should I take for [ disease A ] ?

- Should I take the perscription [ perscription X ] for disease [ disease A ] ?

- Is [ disease A ] discovered within the area [ location A ] ?

But for the purpose of this assignment and simplicity, it only is capable of answering the questions above with the most common diseases and symptoms only. When developed further and data is obtained and maintained by expertise in this field, it can be used to develop a "DoctorBot" application that can diagnose and prescript patient from anywhere at any time for common diseases.

(b) Document in detail (3-4 paragraphs) the steps in your methodology for developing the ontology. What did you find to be the pros and cons of using this strategy?

First, we determined the domain to be symptoms of diseases. Then we thought about the questions the automated question-answering application will answer as listed in part (a). After that, we listed out very common symptoms such as Fever, Nausea, Pain, Tiredness, Diarrhea, Skin Change, etc. Then we discovered some common factors and began to generalizing them into parent classes depending on the visibility of the symptom. From there, we could further specialize some of the classes. For instance, Abdominal Pain, Chest Pain and Headache as subclasses of Pain. After going through the first iteration of symptoms, we started to think about object properties to describe the relationship between symptoms and diseases. We introduced object property such as isSymptomOf and hasSymptom, which inverses each other, to answer the most fundamental questions listed above.

In order to create object property restrictions, we needed another domain of diseases. Similar to the symptom domain process, we first listed out very common diseases such as Flu, Hepatitis, Diabetes, Allergy, Cancer and more. Then we generalized them into some parent classes. Then we realize these disease classes can also have some character / feature / type such as being infectious or non-infectious. So we created a separate class structure Disease Type from Disease since a Disease subclass can be categorized into more than one Disease Type. Then we specialized the Disease Type into the ways of infecting. Is it via air, blood or insects like mosquitos?

As we have the structure of classes, now we needed to put it together and make relationships with object property. We went through the same order as we created the classes from Symptom to Disease and Disease Type. We related Symptom subclasses with Disease with object property isSymptomOf. For example, to say, Cough is a symptom of Flu. Likewise, we related Disease subclasses with Symptom subclasses and DiseaseType subclasses using object property. In order to further answer our competency questions such as "Does [ disease A ] need to have a surgery ?", we introduced data property isSurgeryRequired. To indicate a specific disease that belongs to general Disease subclasses, we then added individuals such as Hepatitis A, Hepatitis B, Lung Cancer, Brain Cancer, etc.

Our ontology development strategy is close to Middle-out strategy, as we started from a core domain, then generalize and specialize as necessary, then branch out and iterate from the beginning again. Since we are starting from some existing data rather than creating a new dataset from the scratch, the middle-out strategy seemed to be the most appropriate. Pros: being flexible while developing in a way that seems the most natural and stable as well as having necessary details. Cons: it may seem inconsistent during the process and it may require frequent refactoring.