Modeling Semantic Search in OO jDREW with an International-Cuisine Taxonomy

by

Group 2

Atteeka Rashid
William Ross
Girish Ranganathan
Bhanu Petchetti

CS6795 Dr. Boley and Dr. Spencer

19 December 2005

0.0 Table of Contents

1.0 Introduction

The goal of our project is to demonstrate how semantic searching might be accomplished. To this end, we have created a mini-web scenario of numerous food items and 24 fictitious restaurants, where each URI relates to one fictitious restaurant's homepage. Furthermore, we have simulated some features of RDF and RDFS using the POSL logic programming representation, and we have developed an international-cuisine taxonomy that we use to classify both restaurants and food dishes. We use the information from our fictitious mini-web, stored as facts in our knowledge base, along with facts and rules inferred by our taxonomy to model the semantic searching of restaurants and food dishes in OO jDREW. Our hope was to gain some experience with tools used in Semantic Web development and to gain some knowledge regarding semantic searching, which we feel is an integral part of the Semantic Web.

2.0 Taxonomy

The following section relates to our *International Cuisine* taxonomy. The taxonomy was initially developed to classify food instances associated with different countries, but, later, an abbreviated version was used to classify restaurants according to country (see Figure 1) and the fuller version was used to keep track of which food instances a particular restaurant serves. As our project is concerned with simulating a mini-web scenario of restaurant websites, we feel that the web crawler should also be considered. Our idea is that web crawlers will eventually become taxonomy specific; that is, there will be some web crawler for each taxonomy. The restaurant-taxonomy web crawler, for example, would use the class names in the taxonomy to match central words in restaurant websites. The address book entry for the word "Mexican," for instance, would point to all Mexican restaurants.

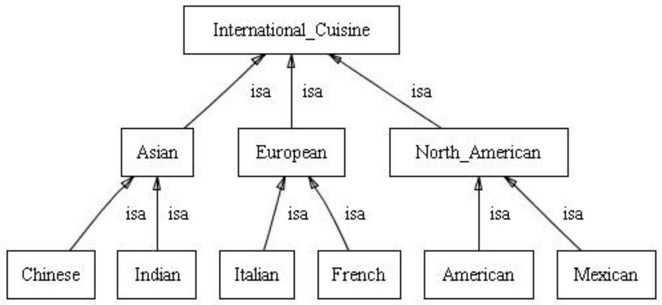


Figure 1: Abbreviated Taxonomy (countries only)

We chose to develop a simplified, international-cuisine taxonomy. We divided foods and restaurants into three continents and further subdivided them into one of the two countries selected for the continent. Finally, we chose representative foods from each country (six to 12 food classes) and modeled the taxonomy using Protégé. The entire taxonomy is comprised of 61 classes. The following figures (Figure 2 to Figure 7) show the food classes that fall under each country. The entire taxonomy was too large to show in its entirety. One final note about the taxonomy: an adjective naming convention is used (e.g., Chinese versus China), since, for example, we say that some food or restaurant is "Chinese," not "China."

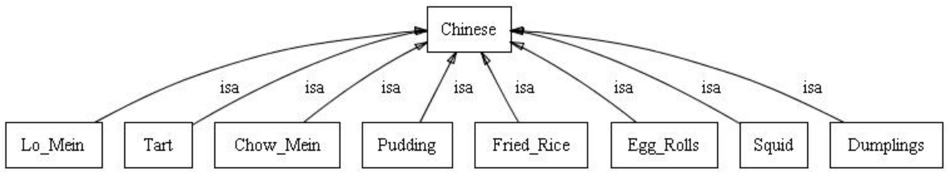


Figure 2: Chinese Food Classes

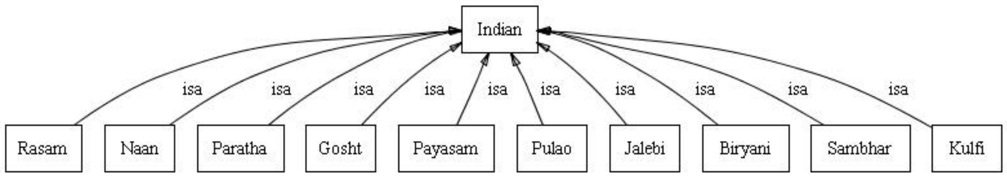


Figure 3: Indian Food Classes

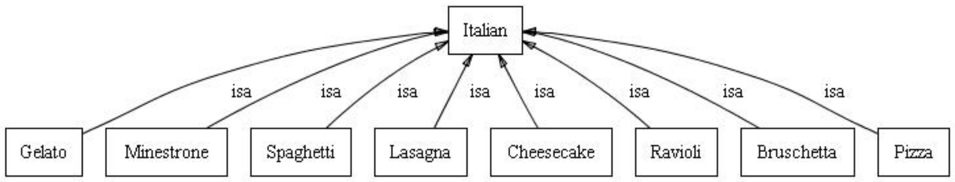


Figure 4: Italian Food Classes

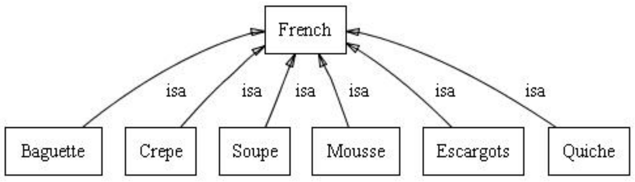


Figure 5: French Food Classes

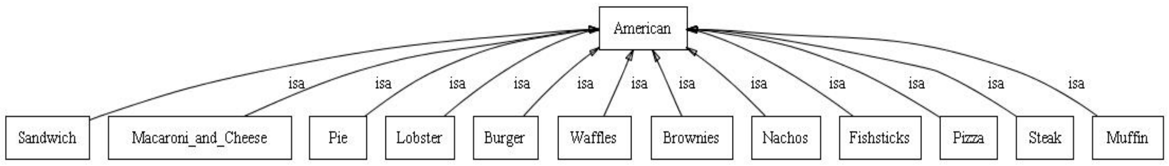


Figure 6: American Food Classes

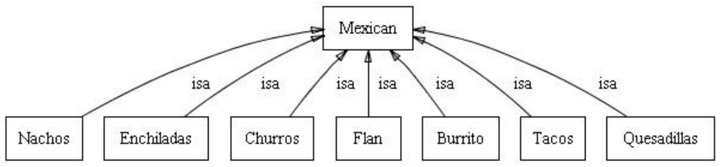


Figure 7: Mexican Food Classes

The dishes under each country are further classified into categories like “breakfast” and “dessert.” However, this information is not captured in the taxonomy; instead, it is specified in our knowledge base using POSL facts and rules (see section 3.0 for more information). The discussion now turns to the tool we used to create our taxonomy.

2.1 Protégé

As mentioned above, we used Protégé to construct our taxonomy. Protégé is a well-known, open-source ontology editor, having many features that can be used when developing knowledge-based applications that depend on some ontology [1]. The program is fairly easy to learn (at least the basics) and allowed us to build and visualize the classes of our taxonomy quickly. In fact, the above figures were obtained using the

OntoViz tab, which gives various views of our taxonomy. We did have trouble getting the OntoViz tab to work correctly on some machines, however, and are not sure why: sometimes the images would appear and other times they would not. Furthermore, we considered using some of Protégé's exporting features (e.g., RDFS and Taxonomic RuleML), but we did not end up using them for our project (see section 6.0 for a more in-depth discussion of the problems we faced and suggested improvements to Protégé, as it relates to exporting). Thus, we were able to use the powerful tool only for its images, although with a few improvements, as mentioned in section 6.0, we believe that Protégé could have aided us a lot. We could have also tried other ontology-development tools like OILED and Ontolingua, but due to time constraints we were not able to experiment with them.

3.0 Knowledge Base

To simulate our mini web, we developed a knowledge base (KB) using POSL (Positional and Slotted Knowledge) notation, following the suggestions in [2]. In our KB, we defined approximately 667 facts and rules,[1] which describe the relationships inferred by our taxonomy (simulating RDFS) and basic property information relating to fictitious food instances and restaurants (simulating RDF); we represent restaurants using simplified Uniform Resource Identifiers (URI's).

[1] See Appendix I for a complete listing of the facts and rules that make up our knowledge base.

3.1 Facts

Our knowledge base contains two categories of facts: (1) those describing the food instances we created for each of the dish classes in our taxonomy (152 facts altogether) and (2) those describing the 24 fictitious restaurants we created to simulate restaurant semantic searching (372 facts altogether). Most of these facts, including all of the food facts, are specified using unary predicates, but some, such as the name of a restaurant (e.g., `name(uri01, "Chez Henri").`), are specified using binary predicates. We will now explain a typical example from each fact category, so that the reader may understand what sort of information is stored in our KB.

The following example demonstrates the facts we have in our KB for the "burrito" dish class. We have two instances: "beef_burrito" and "bean_burrito." Furthermore, we have specified that "bean_burrito" is "vegetarian." This allows us to search for vegetarian dishes and restaurants that serve vegetarian dishes.

```
burrito(beef_burrito).
burrito(bean_burrito).
vegetarian(bean_burrito).
```

In addition to the above example, we have specified two facts in our KB relating to ingredient information: "chocolate(chocolate_mousse)" and "chicken(chicken_burger)." This might be a future extension to our mini web, which could be useful to someone with dietary restrictions (e.g., allergies).

The next example involves the facts stored in our KB for one of our 24 fictitious restaurants. The unary predicate "restaurant" expresses that a certain URI, in this case "uri08," is a restaurant. Each restaurant has its "location," "name," and "star_rating" specified as shown below; these are all binary predicates. Because a subset of the above taxonomy is used to classify restaurants, each restaurant is given a country or continental classification (e.g., one of our restaurants is simply a "european" restaurant). Finally, at least as it relates to facts only relating to restaurants, we use the same "vegetarian" predicate as above to describe a restaurant that serves only vegetarian foods.

```
restaurant(uri08).
location(uri08, "Saint John").
name(uri08, "Martha's Munchies").
star_rating(uri08, 2).
american(uri08).
vegetarian(uri08).
serves(uri08, macaroni_and_cheese).
serves(uri08, whole_wheat_waffles).
serves(uri08, blueberry_waffles).
serves(uri08, blueberry_muffin).
serves(uri08, bran_muffin).
serves(uri08, mixed_fruit_muffin).
serves(uri08, apple_pie).
serves(uri08, blueberry_pie).
serves(uri08, veggie_medley_nachos).
```

One of our most important predicates is "serves." It is used to specify which food instance a particular restaurant serves. For example, the above restaurant serves "bran_muffin," which is an instance of the "muffin" dish class found under "american" in our taxonomy.

3.2 Rules

There are 143 rules in our knowledge base, divided into 3 main categories. The first category involves the rules inferred by our taxonomy, describing the categorization of dishes into continent, nation, and dish classes. For example, an instance of the ravioli dish class is classified by the rules:

```
italian(?X):-ravioli(?X).
european(?X):-italian(?X).
international_cuisine(?X):-european(?X).
```

These rules state that “ravioli” is an “italian” dish, which makes it also a “european” and an “international_cusine” dish. Moreover, they illustrate the RDF Schema (RDFS) feature of subclass relationships between category-types, and some of these rules are even general enough to describe restaurants, too.

The second category consists of additional rules for classifying dishes *independent* of the taxonomy. Included here are rules for classifying some dishes into “main-type” categories we have identified: “pasta,” “soup,” “bread,” “seafood,” “delicacy,” “breakfast,” and “dessert.” For example, the rule `soup(?X):- minestrone(?X)` indicates that a “minestrone” dish should be categorized as belonging to the main-type “soup.” There are also additional rules for defining what objects in the KB are “dishes” (e.g., `dish(?X):- chow_mein(?X)`); these are needed to distinguish a restaurant from a dish, and allow us to search for dishes as well as restaurants.

The third category consists of only a single rule and simulates RDFS’s subPropertyOf. The rule states that if a restaurant serves some dish, and that dish is vegetarian, then the restaurant serves a vegetarian dish. It is specified as follows: `serves_veg(?X, ?Y):-serves(?X, ?Y), vegetarian(?Y).`, and is similar to the example we saw in class about father being a subPropertyOf parent.

3.3 Design Alternatives

A lot of thought went into the final set of facts and rules we decided to store in our KB. For example, one group of facts we did consider storing was ingredient information; as seen above, we do have a couple of general ingredient facts. However, after discussing the matter, we felt that most restaurants would be unwilling to reveal all of the ingredients they use for their dishes ("trade secrets"); hence, we did not include much ingredient information in our KB.

We also thought of using *slot* information to restructure our KB. Here we would have specified a particular food instance as belonging to a particular nation rather than as belonging to a particular dish class. We could have specified "main-type" information using slots as well. This would have substantially reduced the size of our KB. Finally, we could have also eliminated the unary vegetarian predicate. For example,

```
  indian(name->gobi_naan; dish_type->naan; main_type->bread; vegetarian->yes).
```

Another advantage of this design is that it would have alleviated the problems associated with precision, discussed in the next section. However, there is one major drawback to this restructuring: general queries become almost impossible, as dish information is very dependent on nation information. How can one search for all Indian foods, for instance? With more time, we likely could have come up with a more reasonable (and flexible) restructuring. Nonetheless, we did think about using slots and for generality's sake, decided against it.

4.0 Queries

To accomplish semantic searching, machine-understandable information needs to be extracted from websites—in our case, the information relating to our mini-web scenario is stored in our KB. Based on this information, meaningful searches can then be realized by querying the knowledge base. Thus, the quality of semantic searching is highly dependent on the facts and rules contained within the KB. For example, a query for restaurants may return a result if it uses the stored information about restaurants (e.g., location, star rating, country classification, or name); however, a query will fail if it uses any additional information not stored in the KB relating to restaurants (e.g., smoking section information). Furthermore, queries are written in POSL notation, as this is the accepted OO jDREW format, and would be quite cumbersome for "regular" Internet users, who are unfamiliar with logic programming, to write. Despite these drawbacks, our simulation of the semantic search has proven to be quite effective at reducing problems faced by conventional search engines: recall and precision.

If every relevant webpage is returned after issuing the query, then recall is said to be perfect. In our simple mini-web scenario, recall is 100 percent, or perfect. This is reasonable, however, since the query is explicitly written using the facts, rules, and jargon found in our KB. Thus, no issues arise relating to ambiguity such as "Head Pain" versus "Head Ache." In a sense, our queries are comparable to issuing a Prolog query: we expect all possible relevant results.

Precision, on the other hand, is much trickier to improve, as it depends on the amount of detail we put into our KB. For example, consider the following query.

```
serves(?What, meat_pizza), american(meat_pizza).
```

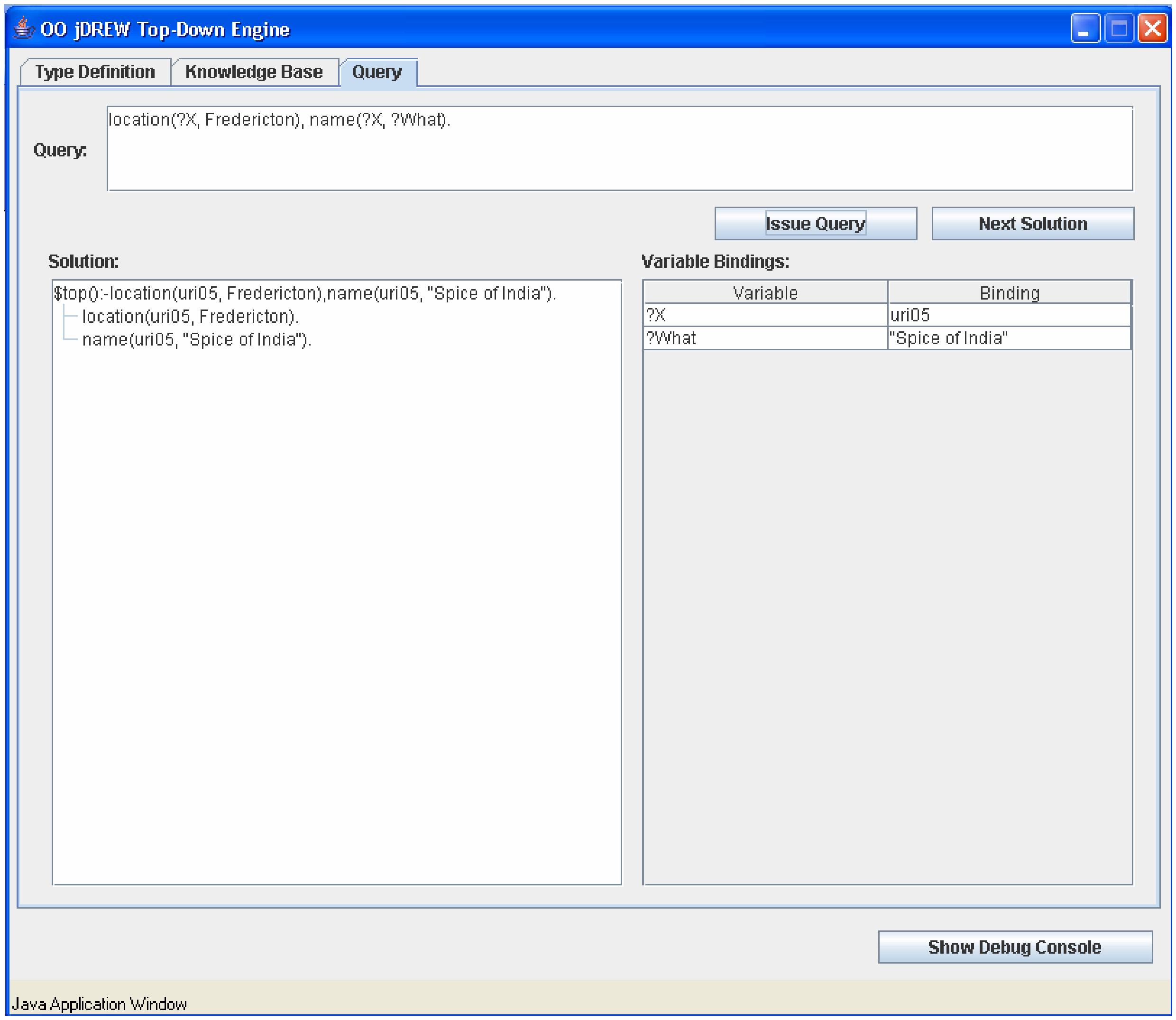
Intuitively, the result of this query should be a list of restaurants that serve American meat pizza. However, the query actually returns any restaurant that serves *either* American or Italian meat pizza. To understand why, let us consider how certain information is stored in the KB. Pizzas can either be American or Italian (in our KB: `american(pizza).` and `italian(pizza).`), and meat pizza is a particular type of pizza (in our KB: `pizza(meat_pizza).`). Also, because of our taxonomy rules, something that is a "pizza" is also an "italian" or an "american" (in our KB: `american(X):- pizza(X).` and `italian(X):- pizza(X).`). Finally, the "serves" relation only maintains information about what food instance a particular restaurant serves (in our KB: `serves(uri--, meat_pizza).`); it does not provide additional information such as what that particular instance's country classification is (e.g., whether it is American or Italian "meat_pizza"). This imprecision in the "serves" relation causes the above query to act counter-intuitively; however, it allows for more general searches like "what restaurants serve meat pizza?"

Precision can be improved, though, by making the information stored in the KB more precise. However, as alluded to in the previous section, there is a price to pay: more precision in the KB may impede more general searches. Nonetheless, recall is still perfect—you get all of the relevant websites and sometimes a couple more. Thus, we feel that the two measures for search results can be substantially improved using semantic searching versus conventional Internet searching.

5.0 Demonstration

Below are three sample search queries, which we feel reflect typical queries a semantic search engine for restaurants would need to perform. They involve querying for restaurant information, as well as dish information, and, we feel, they demonstrate the effectiveness of OO jDREW as a semantic search engine for our mini-web scenario, contained in the KB. The main obstacle in using OO jDREW is the time needed to accomplish some of the more complicated queries (e.g., example query 3 below). However, improvements for making OO jDREW more applicable to widespread semantic searching are suggested in the subsection that follows the three examples.

Example 1: A relatively simple query, in terms of processing time for a result, is to search for the name of any restaurant located in Fredericton. The specification and result of issuing this query is demonstrated in the figure below.



Query: location(?X, Fredericton), name(?X, ?What).

Solution:

```
$top():-location(uri05, Fredericton),name(uri05, "Spice of India").
  location(uri05, Fredericton).
  name(uri05, "Spice of India").
```

Variable Bindings:

Variable	Binding
?X	uri05
?What	"Spice of India"

Figure 8: Easy Example

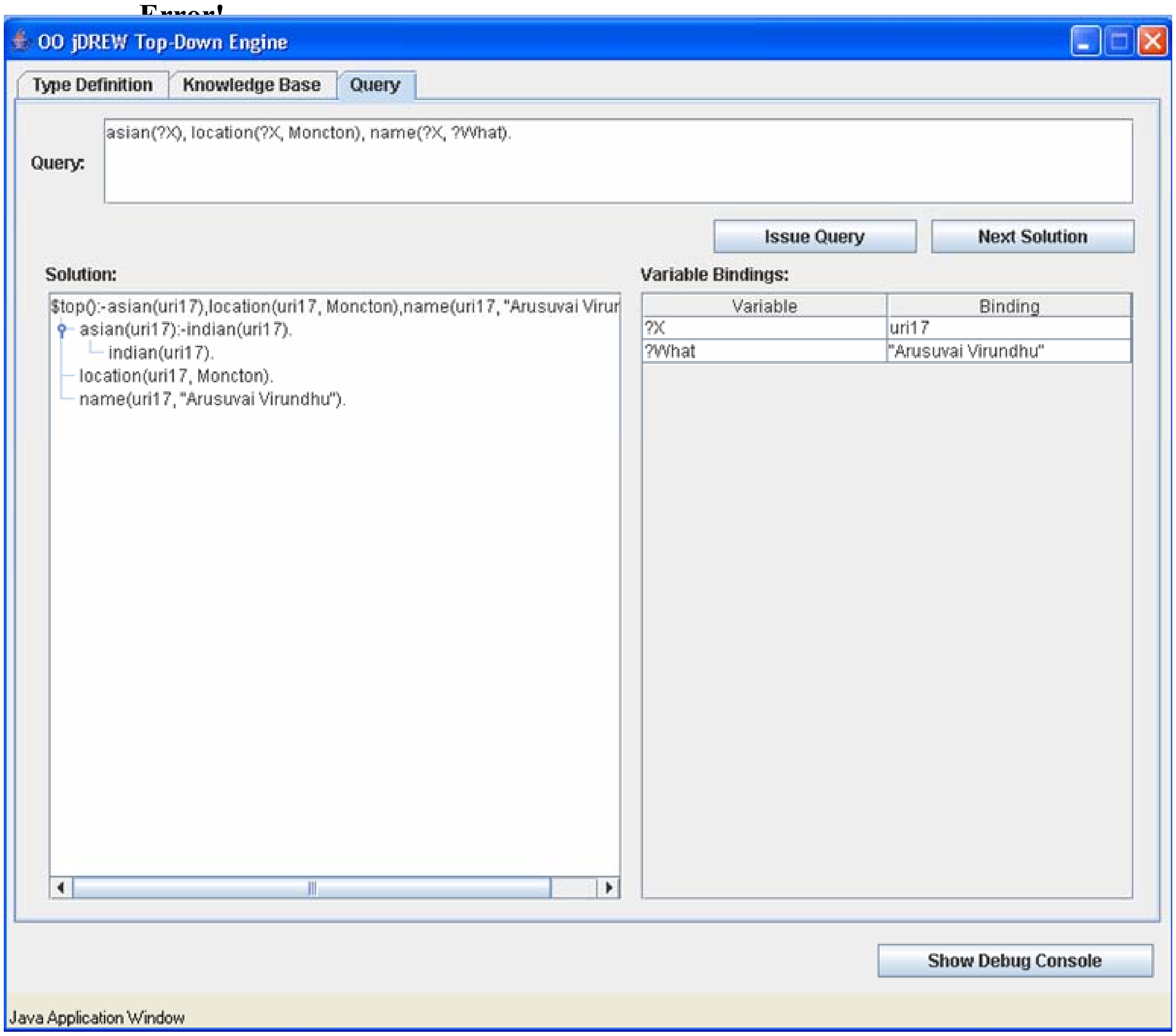
Query: `location(?X,Fredericton), name(?X,?What).`

Output: “Spice of India” (uri05), “Cruisers” (uri06), “Luigi’s Place” (uri07), “Le Diner” (uri10), “Ming’s Marvelous Meals” (uri12), “Tacos Paradise” (uri13), “Latin’s world” (uri15), “Arusuvai Virundhu” (uri17), “Asiano classica” (uri18), “Olive Garden” (uri19), “El Torito” (uri21), “Yet Wah Restaurant” (uri23), and “Khana Khazana” (uri 24).

Result: 100% recall and precision.

Example 2: A more complex query, requiring more processing time, is to search for the name of any Asian restaurant located in Moncton.

Error!



```
asian(?X), location(?X, Moncton), name(?X, ?What).
```

```
$top():-asian(uri17),location(uri17, Moncton),name(uri17, "Arusuvai Virur
  asian(uri17):-indian(uri17).
    indian(uri17).
  location(uri17, Moncton).
  name(uri17, "Arusuvai Virundhu").
```

Variable	Binding
?X	uri17
?What	"Arusuvai Virundhu"

Figure 9: Medium Example

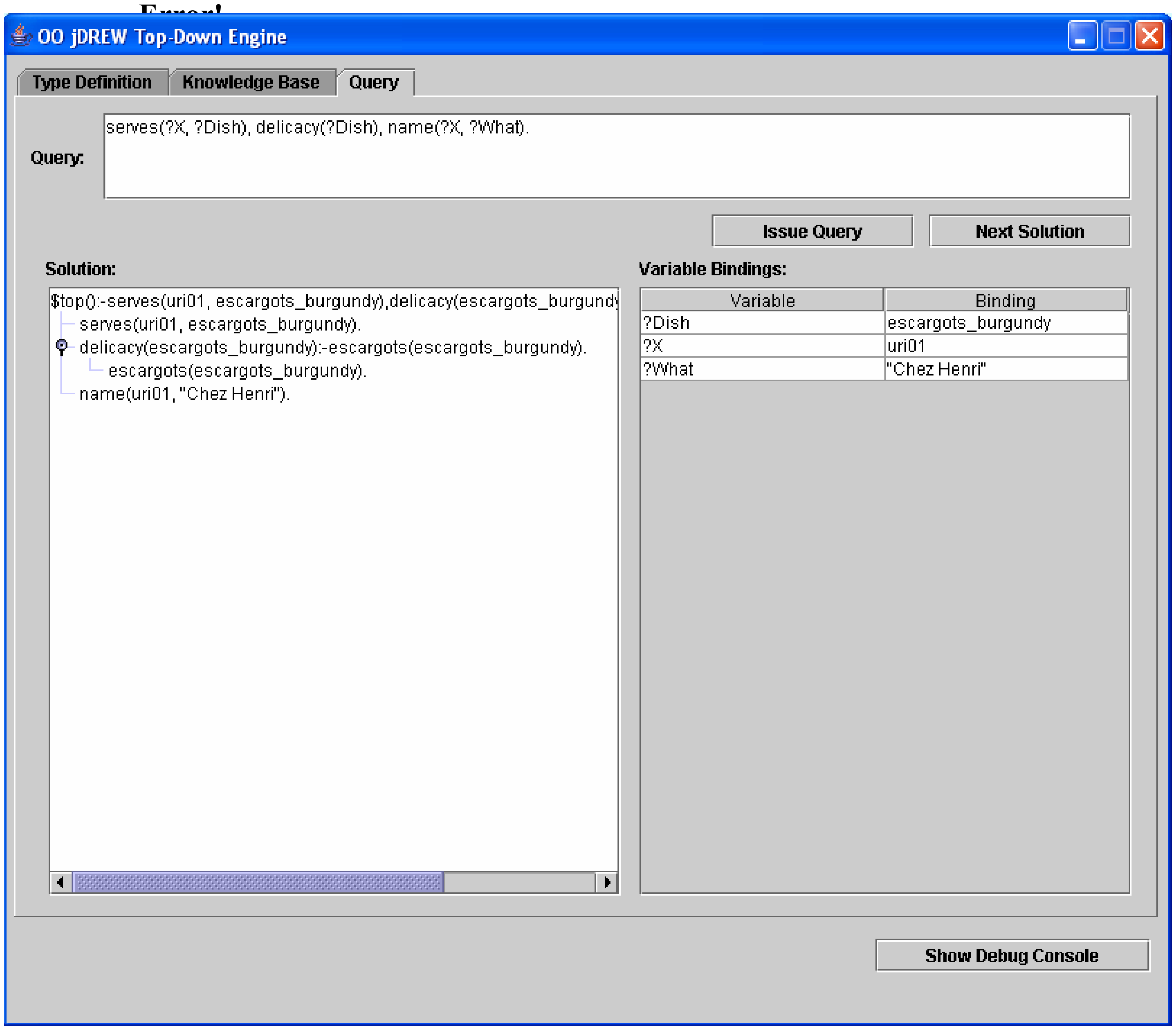
Query: `asian(?X), location(?X,Moncton), name(?X,?What).`

Output: “Arusuvai Virundhu” (uri17) and “Fang’s Palace” (uri02)

Result: 100% recall and precision.

Example 3: A considerably more complex query, requiring a significant amount of time to return a solution, is to search for the name of a restaurant serving a dish that is a delicacy.

Error!



Query: serves(?X, ?Dish), delicacy(?Dish), name(?X, ?What).

Solution:

```
$top():-serves(uri01, escargots_burgundy),delicacy(escargots_burgundy
  serves(uri01, escargots_burgundy).
  delicacy(escargots_burgundy):-escargots(escargots_burgundy).
    escargots(escargots_burgundy).
  name(uri01, "Chez Henri").
```

Variable Bindings:

Variable	Binding
?Dish	escargots_burgundy
?X	uri01
?What	"Chez Henri"

Figure 10: Hard Example

Query: `serves(?X,?Dish), delicacy(?Dish), name(?X, ?What).`

Output: “Chez Henri” (uri01) serves escargots_burgundy, “Frenchz” (uri16) serves escargots_burgundy, “The American Restaurant” (uri20) serves maine_lobster_stew, “La Madeline” (uri22) serves escargots_burgundy, and “Yet Wah Restaurant” serves fried_squid.

Result: 100% recall and precision.

5.1 OO jDREW

To simulate our semantic search engine, we used OO jDREW, which is a reasoning engine developed in Java for performing deductive reasoning on knowledge bases specified in POSL or RuleML [3]. The system is available online [4] in both applet and downloadable formats, and both were used during the course of this project. Moreover, the specific jDREW version we used in our project is the Top-Down, backward-reasoning engine.

We found the Top-Down engine to be very useful and effective in simulating our semantic search, especially in comparison to some other tools available, such as the Bottom-Up version. It allows for the simple specification of numerous *individual* queries, without having to pre-specify the queries as is needed in the bottom-up version, and only outputs the relevant information for that particular query. Furthermore, the derivation tree outputted by OO jDREW helps people to visualize the search process, and the variable bindings outputted, although they are lost once a new query is issued, can be used to refine the search process. Admittedly, since our POSL KB can easily be converted to Prolog, we could have used a system like SWI-Prolog instead. This tool actually does maintain the variable bindings returned by a successful query, so that the variable can be specified in another query by using the "$" prefix in front of the variable name [5]. However, the Top-Down OO jDREW interface provides a much better environment for visualizing and understanding the search process.

Despite its advantages, there are some limitations to the OO jDREW tool. First, for example, while testing subsets of our KB, the parsing on some machines took a very long time (between four and six minutes), while on other machines the parsing took only several seconds. The same sorts of problems occurred when issuing a query. It seems strange that this would occur; thus, a reasonable extension might be to overcome these performance differences (it may have to do with the machines using different versions of Java).

Another problem we encountered was the result of OO jDREW not providing us with error messages when the KB was not entirely parsed. For example, we found that when any comment was missing its comment indicator ("%"), parsing would stop, but no warning message was provided. This created some difficulties when testing queries, as we did not initially understand why some queries were working and others were not. As for the pop-up error messages OO jDREW does provide, although the DebugConsole can help to locate the source of some errors, we feel a useful extension would be to give more information about which line or character caused a particular problem in the pop-up message itself (e.g., missing ")" or "."). For example, we found that if any fact contained a specific type of double-quote character, a message was generated indicating "unexpected char: 0x201C," which did not entirely help us determine the character causing the problem.

Another recognized limitation of the OO jDREW engine is that it is not particularly well suited at processing large KB's. According to [3], it is designed more for small to medium sized KB's. With over 600 rules and facts in our KB, which represents only a small subset of actual international cuisine, we saw a significant time delay when issuing queries requiring a lot of rule processing. Therefore, some extension should be made to allow OO jDREW to process larger, more realistic KB's. It is worth noting that such an extension has already been proposed, involving the use of a more complex data structure to enhance the indexing procedure used in the search [3].

The final extension we will mention is that OO jDREW should be able to dereference our restaurant URI's to actual websites. This, in effect, would allow OO jDREW to be used as a real, Internet, semantic-search engine.

6.0 Alternative Tools

There were other tools, besides those already mentioned, which we considered using for our project. Early on, we did not know which tools were used for which tasks, but we did remember seeing Protégé in the lectures; hence, we began building our taxonomy using this program. Later, we thought that we could use some of Protégé's exporting features—namely, RDF schema (RDFS) and TXRuleML—to facilitate the project implementation; however, we ran into a few troubles on the way, including figuring out how to use type-information within our knowledge base. This section briefly outlines the alternative (sub) tools we thought of using before we adjusted our project description to more closely coincide with that of project description one [6].

As previously mentioned, Protégé is able to represent the taxonomy using RDFS. We thought using this representation would greatly reduce the size of our knowledge base: we would no longer need all of the hierarchical rules. Unfortunately, when Protégé exports RDFS it does so in alphabetical order. OO jDREW seems to need the RDFS to be in the correct hierarchical order to make sense of it. We are not sure why Protégé exports the taxonomy in such a way, but we feel that one useful extension would be to allow for hierarchical RDFS exporting. Manually readjusting the output would take too long, if the taxonomy were quite large. Below are two examples of RDFS that we get from Protégé. The one on the left states that "Asian" is a subclass of "International_Cuisine," whereas the one on the right states that "Bruschetta" is a subclass of "Italian." Both of these facts are expressed in our KB using POSL notation.

```
<owl:Class rdf:ID="Asian">
<rdfs:subClassOf>
<owl:Class rdf:ID="International_Cuisine"/>
</rdfs:subClassOf>
</owl:Class>
```

```
<owl:Class rdf:ID="Bruschetta">
<rdfs:subClassOf>
<owl:Class rdf:ID="Italian"/>
</rdfs:subClassOf>
</owl:Class>
```

We also considered exporting the hierarchy rules from Protégé in RuleML format. The hope was to use the RuleML-to-POSL converter [4] to transform these rules into POSL notation and put them into our KB; this would have saved some typing and allowed us to effectively utilize one of the exporting features. However, there was a slight problem: Protégé exports Taxonomic RuleML (TXRuleML), which is a format neither OO jDREW, nor the converter accept. Therefore, we suggest that either RuleML 0.88 or 0.89 be exported from Protégé or that the converter be modified to convert between POSL and different formats of RuleML, including TXRuleML. Below is a sample conversion between RuleML 0.88 (left) and POSL (right), using the aforementioned converter.

```
<Assert>
   <And>
      <Implies closure="universal">
         <Atom>
            <Rel>mexican</Rel>
            <Var>X</Var>
         </Atom>
         <Atom>
            <Rel>north_american</Rel>
            <Var>X</Var>
         </Atom>
      </Implies>
      <Implies closure="universal">
         <Atom>
            <Rel>french</Rel>
            <Var>X</Var>
         </Atom>
         <Atom>
            <Rel>european</Rel>
            <Var>X</Var>
         </Atom>
      </Implies>
   </And>
</Assert>
```

```
north_american(?X):-mexican(?X).
european(?X):-french(?X).
```

7.0 Conclusion

For our project, we created a knowledge base that contains information about restaurants, dishes, and our international-cuisine taxonomy. Using our fictitious 24-URI mini web and KB, we successfully modeled semantic searching in OO jDREW (the results of which can be seen in section 5.0). Furthermore, in this report, we have addressed the issues of recall and precision, and have stated that both can be improved using semantic searching. However, in order to realize *widespread* semantic searching, there are a couple of issues we feel need to be considered, and both have been concluded from this report.

First, as we have seen, using POSL to write queries is cumbersome. Therefore, what we suggest is that any implementation like ours be used as a backend facility for some real semantic search engine. A more natural-language query could be written in the search engine and, using some sort of ontology (cf. thesaurus), the query could be converted into the jargon of matching taxonomies. A list of relevant taxonomies could then be given, instead of a list of websites, and the entire searching process would become a filtering process. For example, once the user selects the relevant taxonomy, a list of websites could then appear, along with further filtering options (e.g., for a restaurant taxonomy, the restaurants could be filtered by location or star rating).

Some work is being done to enable computers to process English sentences [7]. Using a subset of the English language, Attempto Controlled English (ACE) aims to translate English specifications into a formal specification language, like first-order logic [7]. If this were to succeed, then widespread semantic searching could likely be realized, as ACE could reduce the vagueness and ambiguity inherent in full natural language [7], which is why our queries were specified in POSL, allowing queries to be written more naturally by humans and to be processed more accurately by computers.

The second issue relates to the development of taxonomies. As mentioned earlier in the report, even with our simplified taxonomy there were a lot of design decisions we needed to make. Since we propose universal domain taxonomies (e.g., one taxonomy to fully account for restaurants, or a combination of smaller such taxonomies to create a larger, universal taxonomy that describes restaurants), a lot of cooperation must occur in order to solve concerns like what to model and with how much precision to model items in the taxonomy and KB, so that different taxonomies can be combined. If these two central issues can be resolved (i.e., natural-language query writing and cooperative taxonomy development), we feel confident that semantic searching will overtake conventional Internet searching with great success.

References

1. Stanford Medical Informatics. “The Protégé Ontology Editor and Knowledge Acquisition System.” 2005. The National Center for Biomedical Ontology. National Library of Medicine. 1 Nov. 2005. <http://protege.standford.edu>.
2. Boley, Harold. “Relationships Between Logic Programming and RDF.” <http://www.dfki.uni-kl.de/~boley/rdfptalk.pdf>.
3. Ball, Marcel. “OO jDREW: Design and Implementation of a Reasoning Engine for the Semantic Web.” CS4997 Honours Thesis Project Report, UNB. 6 April 2005. <http://www.jdrew.org/oojdrew/docs/OOjDREW.pdf>.
4. Ball, Marcel. “OO jDREW.” 27 July 2005. 1 Nov. 2005. <http://www.jdrew.org/oojdrew/>.
5. Wielemaker, Jan. “SWI-Prolog 5.5 Reference Manual.” University of Amsterdam (2005): 13. 5 Nov. 2005. <http://wwwtcs.inf.tu-dresden.de/~nauber/refman5_5.pdf#search='SWI%20Prolog%20Reference%20Guide%205.5'>.
6. Spencer, Bruce and Harold Boley. “Project 1: Modelling Semantic Search in j-DREW.” Dec. 2005. CS 6795 Semantic Web Techniques. 31 Oct. 2005. <http://www.cs.unb.ca/~bspencer/cs6795swt/project1.html>.
7. Fuchs, Norbert E. “Attempto Controlled English: A Specification and Knowledge Representation Language.” 13 July 2004. University of Zurich. 2 Dec. 2005. <http://www.ifi.unizh.ch/attempto/description/index.html>.

Appendix I: Knowledge Base

%**The dish rules**:

```
% A dish is international cuisine if it is North American, or European, or Asian:
international_cuisine(?X):-north_american(?X).
international_cuisine(?X):-european(?X).
international_cuisine(?X):-asian(?X).

% A dish is North American if it is American, or Mexican:
north_american(?X):-american(?X).
north_american(?X):-mexican(?X).
european(?X):-french(?X).
european(?X):-italian(?X).
asian(?X):-indian(?X).
asian(?X):-chinese(?X).

% Dish nationality rules:
american(?X):-pizza(?X).
american(?X):-sandwich(?X).
american(?X):-fish_sticks(?X).
american(?X):-lobster(?X).
american(?X):-steak(?X).
american(?X):-burger(?X).
american(?X):-macaroni_and_cheese(?X).
american(?X):-nachos(?X).
american(?X):-muffin(?X).
american(?X):-waffles(?X).
american(?X):-pie(?X).
american(?X):-brownies(?X).

mexican(?X):-tacos(?X).
mexican(?X):-enchiladas(?X).
mexican(?X):-burrito(?X).
mexican(?X):-quesadillas(?X).
mexican(?X):-nachos(?X).
mexican(?X):-flan(?X).
mexican(?X):-churros(?X).

french(?X):-quiche(?X).
french(?X):-baguette(?X).
french(?X):-soupe(?X).
french(?X):-escargots(?X).
french(?X):-crepe(?X).
french(?X):-mousse(?X).

italian(?X):-pizza(?X).
italian(?X):-spaghetti(?X).
italian(?X):-lasagna(?X).
italian(?X):-ravioli(?X).
italian(?X):-bruschetta(?X).
italian(?X):-minestrone(?X).
italian(?X):-gelato(?X).
italian(?X):-cheesecake(?X).

indian(?X):-gosht(?X).
indian(?X):-naan(?X).
indian(?X):-biryani(?X).
indian(?X):-paratha(?X).
indian(?X):-payasam(?X).
indian(?X):-pulao(?X).
indian(?X):-rasam(?X).
indian(?X):-sambhar(?X).
indian(?X):-kulfi(?X).
indian(?X):-jalebi(?X).

chinese(?X):-chow_mein(?X).
chinese(?X):-lo_mein(?X).
chinese(?X):-egg_rolls(?X).
chinese(?X):-fried_rice(?X).
chinese(?X):-dumplings(?X).
chinese(?X):-squid(?X).
chinese(?X):-pudding(?X).
chinese(?X):-tart(?X).
```

% Dish **'Main Type'** category rules:

```
pasta(?X):-macaroni_and_cheese(?X).
pasta(?X):-spaghetti(?X).
pasta(?X):-lasagna(?X).
pasta(?X):-ravioli(?X).

soup(?X):-soupe(?X).
soup(?X):-minestrone(?X).

bread(?X):-baguette(?X).
bread(?X):-bruschetta(?X).
bread(?X):-naan(?X).

seafood(?X):-fish_sticks(?X).
seafood(?X):-lobster(?X).
seafood(?X):-escargots(?X).
seafood(?X):-squid(?X).

delicacy(?X):-lobster(?X).
delicacy(?X):-escargots(?X).
delicacy(?X):-squid(?X).

breakfast(?X):-muffin(?X).
breakfast(?X):-waffle(?X).
breakfast(?X):-crepe(?X).

dessert(?X):-muffin(?X).
dessert(?X):-waffle(?X).
dessert(?X):-pie(?X).
dessert(?X):-brownie(?X).
dessert(?X):-flan(?X).
dessert(?X):-churros(?X).
dessert(?X):-crepe(?X).
dessert(?X):-mousse(?X).
dessert(?X):-gelato(?X).
dessert(?X):-cheesecake(?X).
dessert(?X):-kulfi(?X).
dessert(?X):-jalebi(?X).
dessert(?X):-pudding(?X).
dessert(?X):-tart(?X).

dish(?X):-pizza(?X).              %american
dishes
dish(?X):-sandwich(?X).
dish(?X):-fish_sticks(?X).
dish(?X):-lobster(?X).
dish(?X):-steak(?X).
dish(?X):-burger(?X).
dish(?X):-macaroni_and_cheese(?X).
dish(?X):-nachos(?X).
dish(?X):-muffin(?X).
dish(?X):-waffles(?X).
dish(?X):-pie(?X).
dish(?X):-brownies(?X).
```

```
dish(?X):-tacos(?X).            %mexican dishes
dish(?X):-enchiladas(?X).
dish(?X):-burrito(?X).
dish(?X):-quesadillas(?X).
dish(?X):-flan(?X).
dish(?X):-churros(?X).
dish(?X):-quiche(?X).           %french dishes
dish(?X):-baguette(?X).
dish(?X):-soupe(?X).
dish(?X):-escargots(?X).
dish(?X):-crepe(?X).
dish(?X):-mousse(?X).
dish(?X):-spaghetti(?X).        %italian dishes
dish(?X):-lasagna(?X).
dish(?X):-ravioli(?X).
dish(?X):-bruschetta(?X).
dish(?X):-minestrone(?X).
dish(?X):-gelato(?X).
dish(?X):-cheesecake(?X).
dish(?X):-gosht(?X).            %indian dishes
dish(?X):-naan(?X).
dish(?X):-biryani(?X).
dish(?X):-paratha(?X).
dish(?X):-payasam(?X).
dish(?X):-pulao(?X).
dish(?X):-rasam(?X).
dish(?X):-sambhar(?X).
dish(?X):-kulfi(?X).
dish(?X):-jalebi(?X).
dish(?X):-chow_mein(?X).        %chinese dishes
dish(?X):-lo_mein(?X).
dish(?X):-egg_rolls(?X).
dish(?X):-fried_rice(?X).
dish(?X):-dumplings(?X).
dish(?X):-squid(?X).
dish(?X):-pudding(?X).
dish(?X):-tart(?X).
```

```
% The dish facts:

% American dish instances
sandwich(omelet_sandwich).
sandwich(vegetable_sandwich).
vegetarian(vegetable_sandwich).
sandwich(chicken_sandwich).
sandwich(egg_salad_sandwich).

fish_sticks(fish_sticks).

burger(burger_and_fries).
burger(hamburger).
burger(veggie_burger).
vegetarian(veggie_burger).
burger(cheese_burger).

muffin(blueberry_muffin).
muffin(bran_muffin).
muffin(mixed_fruit_muffin).

waffles(whole_wheat_waffles).
waffles(blueberry_waffles).

brownies(caramel_turtle_brownies).
brownies(chestnut_truffle_brownies).

steak(flaming_steak_diane).
steak(pan_fried_rib_steak).

lobster(lobster_chowder).
lobster(maine_lobster_stew).

macaroni_and_cheese(macaroni_and_c
heese).

pie(apple_pie).
pie(blueberry_pie).
pie(lemon_meringue_pie).

nachos(blue_corn_nachos).
nachos(veggie_medley_nachos).
vegetarian(veggie_medley_nachos).

pizza(meat_pizza).
pizza(veg_pizza).
vegetarian(veg_pizza).


% Mexican dish instances:
tacos(garlic_tacos).
tacos(beef_tacos).
tacos(tofu_tacos).
vegetarian(tofu_tacos).
enchiladas(beef_enchiladas).
enchiladas(chicken_enchiladas).
enchiladas(breakfast_enchiladas).
vegetarian(breakfast_enchiladas).
breakfast(breakfast_enchiladas).

burrito(bean_burrito).
burrito(beef_burrito).
vegetarian(bean_burrito).

quesadillas(cheese_quesadillas).
quesadillas(chicken_quesadillas).

nachos(veggie_medley_nachos).
nachos(cheese_nachos).
vegetarian(veggie_medley_nachos).

flan(caramel_drizzled_flan).
flan(chocolate_flan).

churros(apple_churros).


% French dish instances:
mousse(chocolate_mousse).
mousse(strawberry_mousse).
mousse(vanilla_mousse).

crepe(breakfast_crepe).
crepe(chocolate_crepe).
crepe(veg_crepe).
vegetarian(veg_crepe).

quiche(quiche_lorraine).
quiche(veg_quiche).
vegetarian(veg_quiche).

baguette(baguette).
vegetarian(baguette).

soupe(soupe_oignon).
soupe(soupe_lentilles).
vegetarian(soupe_lentilles).

escargots(escargots_burgundy).


% Italian dish instances:
pizza(pineapple_pizza).
pizza(chicken_pizza).
pizza(anchovey_pizza).
pizza(olive_pizza).
pizza(meat_pizza).
vegetarian(pineapple_pizza).
seafood(anchovey_pizza).
```

```
spaghetti(meatball_spaghetti).
spaghetti(spinach_spaghetti).
vegetarian(spinach_spaghetti).

lasagna(three_cheese_lasagna).
lasagna(beef_lasagna).
vegetarian(three_cheese_lasagna).

ravioli(cheese_ravioli).
ravioli(beef_ravioli).

brushetta(garlic_bruschetta).
bruchetta(cheese_topped_bruschetta).
vegetarian(garlic_bruschetta).
vegetarian(cheese_topped_bruschetta).

minestrone(veggie_medley_minestrone).
minestrone(cheese_topped_minestrone).
vegetarian(veggie_medley_minestrone).
vegetarian(cheese_topped_minestrone).

gelato(vanilla_gelato).
gelato(strawberry_gelato).
gelato(chocolate_gelato).

cheesecake(chocolate_cheesecake).
cheesecake(cherry_cheesecake).


% Indian dish instances:
paratha(allu_paratha).
vegetarian(allu_paratha).
paratha(shahi_paratha).

naan(gobi_naan).
vegetarian(gobi_naan).

gosht(masala_gosht).
vegetarian(masala_gosht).
gosht(mutton_masala_gosht).
gosht(gosht_e_akbari).
vegetarian(gosht_e_akbari).
gosht(chicken_gosht_e_akbari).

biryani(murgh_dum_biryani).
vegetarian(murgh_dum_biryani).
biryani(meat_murgh_dum_biryani).
biryani(yakhni_biryani).
vegetarian(yakhni_biryani).
biryani(beef_yakhni_biryani).

pulao(mattar_pulao).
vegetarian(mattar_pulao).
pulao(panchrangi_pulao).
vegetarian(panchrangi_pulao).

rasam(pepper_rasam).
vegetarian(pepper_rasam).
rasam(tomato_rasam).
vegetarian(tomato_rasam).

sambhar(kadamba_sambhar).
vegetarian(kadamba_sambhar).
sambhar(chicken_sambhar).

payasam(pal_payasam).
payasam(semiya_payasam).

kulfi(mango_kulfi).
kulfi(almond_kulfi).

jalebi(jalebi).


% Chinese dish instances:
chow_mein(chicken_chow_mein).
chow_mein(veg_chow_mein).
vegetarian(veg_chow_mein).

lo_mein(beef_lo_mein).
lo_mein(veg_lo_mein).
vegetarian(veg_lo_mein).

squid(fried_squid).
squid(squid_balls).

dumplings(pork_dumplings).
dumplings(vegan_dumplings).
vegetarian(vegan_dumplings).

fried_rice(chicken_fried_rice).
fried_rice(veg_fried_rice).
vegetarian(veg_fried_rice).

pudding(mango_pudding).
pudding(coconut_pudding).

tart(sago_tart).
tart(custard_tart).

egg_rolls(meat_egg_roll).
egg_rolls(veg_egg_roll).
vegetarian(veg_egg_roll).


% Sample facts for searching for
ingredient information:
chicken(chicken_burger).
chocolate(chocolate_mousse).
```

```
% The restaurant facts:

% French-Italian Restaurant
restaurant(uri01).
location(uri01, Moncton).
name(uri01, "Chez Henri").
star_rating(uri01, 4).
european(uri01).
serves(uri01, chocolate_mousse).
serves(uri01, strawberry_mousse).
serves(uri01, veg_crepe).
serves(uri01, veg_crepe).
serves(uri01,breakfast_crepe).
serves(uri01, escargots_burgundy).
serves(uri01, baguette).
serves(uri01, quiche_lorraine).
serves(uri01, soupe_lentilles).
serves(uri01,
veggie_medley_minestrone).
serves(uri01, garlic_brushetta).
serves(uri01, vanilla_gelato).
serves(uri01, three_cheese_lasagna).
serves(uri01, beef_lasagna).

% Chinese Restaurant
restaurant(uri02).
location(uri02, Moncton).
name(uri02, "Fang's Palace").
star_rating(uri02, 3).
chinese(uri02).
serves(uri02, veg_lo_mein).
serves(uri02, veg_lo_mein).
serves(uri02, veg_chow_mein).
serves(uri02, chicken_chow_mein).
serves(uri02, pork_dumplings).
serves(uri02, veg_fried_rice).
serves(uri02, chicken_fried_rice).
serves(uri02, veg_egg_roll).
serves(uri02, mango_pudding).
serves(uri02, sago_tart).

% Mexican Restaurant
restaurant(uri03).
location(uri03, "Saint John").
name(uri03, "El Taco").
star_rating(uri03, 2).
mexican(uri03).
serves(uri03, beef_tacos).
serves(uri03, tofu_tacos).
serves(uri03, chicken_quesadillas).
serves(uri03, cheese_quesadillas).
serves(uri03, bean_burrito).
serves(uri03, beef_burrito).
serves(uri03, cheese_nachos).
serves(uri03, chocolate_flan).

% Italian Restaurant
restaurant(uri04).
location(uri04, Moncton).
name(uri04, "Papas Pizza").
star_rating(uri04, 3).
italian(uri04).
serves(uri04, pineapple_pizza).
serves(uri04, anchovy_pizza).
serves(uri04, chicken_pizza).
serves(uri04, olive_pizza).
serves(uri04, meat_pizza).
serves(uri04, meatball_spaghetti).
serves(uri04, three_cheese_lasagna).
serves(uri04, beef_lasagna).
serves(uri04, cheese_ravioli).
serves(uri04, beef_ravioli).
serves(uri04, chocolate_cheesecake).
serves(uri04, cherry_cheesecake).

% Indian Restaurant
restaurant(uri05).
location(uri05, Fredericton).
name(uri05, "Spice of India").
star_rating(uri05, 3).
indian(uri05).
serves(uri05, allu_paratha).
serves(uri05, shahi_paratha).
serves(uri05, masala_gosht).
serves(uri05, tomato_rasam).
serves(uri05, chicken_sambhar).
serves(uri05, mango_kulfi).

% American Restaurant
restaurant(uri06).
location(uri06, Fredericton). %At multiple
locations
location(uri06, Moncton).
location(uri06, "Saint John").
name(uri06, Cruisers).
star_rating(uri06, 2).
american(uri06).
serves(uri06, meat_pizza).
serves(uri06, burger_and_fries).
serves(uri06, veg_pizza).
serves(uri06, fish_sticks).
serves(uri06, apple_pie).
serves(uri06, macaroni_and_cheese).
```

```
% Italian-American restaurant:
restaurant(uri07).
location(uri07, Fredericton).
name(uri07, "Luigi's Place").
star_rating(uri07, 3).
italian(uri07).
american(uri07).
serves(uri07, meat_pizza).
serves(uri07, chicken_pizza).
serves(uri07, spinach_spaghetti).
serves(uri07, beef_lasagna).
serves(uri07,
cheese_topped_bruschetta).
serves(uri07, beef_ravioli).
serves(uri07, veg_pizza). % american
serves(uri07, veggie_burger).
serves(uri07, cheese_burger).
serves(uri07, chicken_sandwich).
serves(uri07, lemon_meringue_pie).
serves(uri07, strawberry_gelato).
serves(uri07, chocolate_cheesecake).

% American restaurant (vegetarian
breakfast place):
restaurant(uri08).
location(uri08, "Saint John").
name(uri08, "Martha's Munchies").
star_rating(uri08, 2).
american(uri08).
vegetarian(uri08).
serves(uri08, macaroni_and_cheese).
serves(uri08, whole_wheat_waffles).
serves(uri08, blueberry_waffles).
serves(uri08, blueberry_muffin).
serves(uri08, bran_muffin).
serves(uri08, mixed_fruit_muffin).
serves(uri08, apple_pie).
serves(uri08, blueberry_pie).
serves(uri08, veggie_medley_nachos).

% Mexican restaurant:
restaurant(uri09).
location(uri09, Moncton).
name(uri09, "Amigos").
star_rating(uri09, 3).
mexican(uri09).
serves(uri09, garlic_tacos).
serves(uri09, tofu_tacos).
serves(uri09, beef_enchiladas).
serves(uri09, bean_burrito).
serves(uri09, chicken_quesadillas).
serves(uri09, veggie_medley_nachos).
serves(uri09, caramel_drizzled_flan).
serves(uri09, apple_churros).

% French restaurant:
restaurant(uri10).
location(uri10, Fredericton).
name(uri10, "Le Diner").
star_rating(uri10, 2).
french(uri10).
serves(uri10, breakfast_crepe).
serves(uri10, veg_crepe).
serves(uri10, veg_quiche).
serves(uri10, soupe_oignon).
serves(uri10, soupe_lentilles).
serves(uri10, baguette).
serves(uri10, vanilla_mousse).

% Indian restaurant:
restaurant(uri11).
location(uri11, "Saint John").
name(uri11, "Babul Biryani House").
star_rating(uri11, 4).
indian(uri11).
serves(uri11, murgh_dum_biryani).
serves(uri11, yakhni_biryani).
serves(uri11, shahi_paratha).
serves(uri11, gobi_naan).
serves(uri11, gosht_e_akbari).
serves(uri11, kadamba_sambhar).
serves(uri11, chicken_sambhar).
serves(uri11, pal_payasam).
serves(uri11, mango_kulfi).
serves(uri11, almond_kulfi).
serves(uri11, jalebi).

% Chinese restaurant:
restaurant(uri12).
location(uri12, Fredericton).
name(uri12, "Ming's Marvelous Meals").
star_rating(uri12, 3).
chinese(uri12).
serves(uri12, beef_lo_mein).
serves(uri12, chicken_chow_mein).
serves(uri12, veg_chow_mein).
serves(uri12, vegan_dumplings).
serves(uri12, chicken_fried_rice).
serves(uri12, veg_fried_rice).
serves(uri12, sago_tart).
serves(uri12, custard_tart).
```

```
% Mexican Restaurant
restaurant(uri13).
location(uri13, Fredericton).
name(uri13, "Tacos Paradise").
star_rating(uri13, 4).
mexican(uri13).
serves(uri13, tofu_tacos).
serves(uri13, garlic_tacos).
serves(uri13, beef_enchiladas).
serves(uri13, breakfast_enchiladas).
serves(uri13, cheese_quesadillas).
serves(uri13, bean_burrito).
serves(uri13, beef_burrito).
serves(uri13, veggie_medley_nachos).
serves(uri13, cheese_nachos).
serves(uri13, apple_churros).
serves(uri13, caramel_drizzled_flan).

% Italian Restaurant
restaurant(uri14).
location(uri14, "Saint John").
name(uri14, "Pizza world").
star_rating(uri14, 3).
italian(uri14).
serves(uri14, pineapple_pizza).
serves(uri14, anchovy_pizza).
serves(uri14, olive_pizza).
serves(uri14, meat_pizza).
serves(uri14, meatball_spaghetti).
serves(uri14, three_cheese_lasagna).
serves(uri14, beef_lasagna).
serves(uri14, cheese_ravioli).
serves(uri14, garlic_bruschetta).
serves(uri14,
veggie_medley_minestrone).
serves(uri14, vanilla_gelato).
serves(uri14, strawberry_gelato).
serves(uri14, chocolate_cheesecake).

% American Restaurant
restaurant(uri15).
location(uri15, Fredericton).
name(uri15, "Latin's world").
star_rating(uri15, 4).
american(uri15).
serves(uri15, omelet_sandwich).
serves(uri15, vegetable_sandwich).
serves(uri15, hamburger).
serves(uri15, blueberry_muffin).
serves(uri15, bran_muffin).
serves(uri15, caramel_turtle_brownies).
serves(uri15, flaming_steak_diane).
serves(uri15, apple_pie).
serves(uri15, blue_corn_nachos).
serves(uri15, meat_pizza).
serves(uri15, veg_pizza).

% French Restaurant
restaurant(uri16).
location(uri16, Moncton).
name(uri16, "Frenchz").
star_rating(uri16, 3).
french(uri16).
serves(uri16, chocolate_mousse).
serves(uri16, vanilla_mousse).
serves(uri16, breakfast_crepe).
serves(uri16, quiche_lorraine).
serves(uri16, veg_quiche).
serves(uri16, soupe_oignon).
serves(uri16, soupe_lentilles).
serves(uri16, escargots_burgundy).

% Indian Restaurant
restaurant(uri15).
location(uri17, Fredericton).
location(uri17, Moncton).
name(uri17, "Arusuvai Virundhu").
star_rating(uri17, 4).
indian(uri17).
serves(uri17, allu_paratha).
serves(uri17, shahi_paratha).
serves(uri17, masala_gosht).
serves(uri17, tomato_rasam).
serves(uri17, chicken_sambhar).
serves(uri17, mutton_masala_gosht).
serves(uri17, gosht_e_akbari).
serves(uri17, chicken_gosht_e_akbari).
serves(uri17, yakhni_biryani).
serves(uri17, meat_murgh_dum_biryani).
serves(uri17, beaf_yakhni_biryani).
serves(uri17, mattar_pulao).
serves(uri17, pal_payasam).
serves(uri17, semiya_payasam).
serves(uri17, mango_kulfi).
serves(uri17, almond_kulfi).

% Asian Restaurant
restaurant(uri18).
location(uri18, Fredericton).
name(uri18, "Asiano classica").
star_rating(uri18, 4).
asian(uri18).
serves(uri18, allu_paratha).
serves(uri18, veg_chow_mein).
serves(uri18, shahi_paratha).
serves(uri18, chicken_chow_mein).
serves(uri18, masala_gosht).
serves(uri18, pork_dumplings).
serves(uri18, tomato_rasam).
serves(uri18, veg_fried_rice).
serves(uri18, chicken_sambhar).
serves(uri18, chicken_fried_rice).
serves(uri18, almond_kulfi).
serves(uri18, veg_egg_roll).
serves(uri18, mango_pudding).
serves(uri18, custard_tart).

% Italian restaurant:
restaurant(uri19).
location(uri19, Fredericton).
name(uri19, "Olive Garden").
star_rating(uri19, 4).
italian(uri19).
serves(uri19, olive_pizza).
serves(uri19, meat_pizza).
serves(uri19, anchovey_pizza).
serves(uri19, spinach_spaghetti).
serves(uri19, beef_lasagna).
serves(uri19, three_cheese_lasagna).
serves(uri19, beef_ravioli).
serves(uri19, garlic_bruschetta).
serves(uri19,
cheese_topped_minestrone).
serves(uri19, strawberry_gelato).
serves(uri19, cherry_cheesecake).

%  American restaurant:
restaurant(uri20).
location(uri20, Miramachi).
name(uri20, "The American Restaurant").
star_rating(uri20, 3).
american(uri20).
serves(uri20, egg_salad_sandwich).
serves(uri20, vegetable_sandwich).
serves(uri20, burger_and_fries).
serves(uri20, fish_sticks).
serves(uri20, cheese_burger).
serves(uri20, blueberry_muffin).
serves(uri20, whole_wheat_waffles).
serves(uri20, caramel_turtle_brownies).
serves(uri20, flaming_steak_diane).
serves(uri20, maine_lobster_stew).
serves(uri20, blueberry_pie).
serves(uri20, meat_pizza).
serves(uri20, veg_pizza).

% Mexican restaurant:
restaurant(uri21).
location(uri21, Fredericton).
name(uri21, "El Torito").
star_rating(uri21, 2).
mexican(uri21).
serves(uri21, tofu_tacos).
serves(uri21, beef_tacos).
serves(uri21, chicken_enchiladas).
serves(uri21, breakfast_enchiladas).
serves(uri21, bean_burrito).
serves(uri21, chicken_quesadillas).
serves(uri21, cheese_nachos).
serves(uri21, chocolate_flan).% desert
serves(uri21, apple_churros).

%  French restaurant:
restaurant(uri22).
location(uri22, Moncton).
name(uri22, "La Madeline").
star_rating(uri22, 3).
french(uri22).
serves(uri22, chocolate_mousse).
serves(uri22, vanilla_mousse).
serves(uri22, breakfast_crepe).
serves(uri22, veg_crepe).
serves(uri22, baguette).
serves(uri22, soupe_lentilles).
serves(uri22, escargots_burgundy).

% Chinese restaurant:
restaurant(uri23).
location(uri23, Fredericton).
name(uri23, "Yet Wah Restaurant").
star_rating(uri23, 4).
chinese(uri23).
serves(uri23, veg_chow_mein).
serves(uri23, beef_lo_mein).
serves(uri23, fried_squid).  %seafood
serves(uri23, vegan_dumplings).
serves(uri23, pork_dumplings).
serves(uri23, chicken_fried_rice).
serves(uri23, veg_fried_rice).
serves(uri23, custard_tart).
serves(uri23, coconut_pudding).
serves(uri23, meat_egg_roll).

% Indian Restaurant
restaurant(uri24).
location(uri24, Fredericton).
name(uri24, "Khana Khazana").
star_rating(uri24, 4).
indian(uri24).
serves(uri24, shahi_paratha).
serves(uri24, gobi_naan).
serves(uri24, mutton_masala_gosht).
serves(uri24, chicken_gosht_e_akbari).
serves(uri24, murgh_dum_biryani).
serves(uri24, yakhni_biryani).
serves(uri24, mattar_pulao).
serves(uri24, panchrangi_pulao).
serves(uri24, pepper_rasam).
serves(uri24, kadamba_sambhar).
serves(uri24, mango_kulfi).
serves(uri24, jalebi).

% A rule simulating RDFS's
subPropertyOf:
serves_veg(?X, ?Y):-serves(?X, ?Y),
vegetarian(?Y).
```