Using Ontologies to Derive Transitive Queries

Graham J.L. Kemp
Department of Computing Science
Chalmers University of Technology
SE-412 96, Göteborg, Sweden
kemp@cs.chalmers.se

Implementing ontologies in a database has several advantages, including providing a framework within which the contents of the ontology can be queried, and integrity checks can be made. However, when both the hierarchical vocabulary of terms and the data described by these terms are stored in a database then there are possibilities to use the ontology automatically to derive transitive queries based on queries asked by users. The work described here makes use of the P/FDM database management system[1], and the schema extracts and queries given here are expressed in the Daplex language.

The first example shows how the NCBI taxonomy[2] can be used when answering a user query about samples in a database of microarray-based gene expression data. An extract from the schema for that database is shown here:

```
declare sample                       ->> entity
  declare id(sample)                  -> string
  declare organism_ncbi(sample)       -> string
  declare treatment_protocol(sample) -> string
  declare used_in_experiment(sample) -> experiment
  declare sample_parameters(sample) ->> parameter
  key_of sample is id
```

Terms from the NCBI taxonomy are stored in a separate database module. An extract from the schema for that module is shown here:

```
declare taxonomy_entry               ->> entity
  declare name(taxonomy_entry)        -> string
  declare is_of_taxa(taxonomy_entry) -> taxonomy_entry
  key_of taxonomy_entry is name
```

The relationship `is_of_taxa` relates a term in the taxonomy to the super-term immediately above it. The following user query asks for information about samples where the term "rodentia" describes the sample:

```
for each s in sample such that organism_ncbi(s)="rodentia"
  print(id(s),treatment_protocol(s));
```

[1] http://www.csd.abdn.ac.uk/~pfdm/
[2] http://www.ncbi.nlm.nih.gov/Taxonomy/

It is possible that more specialised terms, below "rodentia" in the taxonomy tree, will be recorded with some samples rather than the name "rodentia" itself. If a user asks for information about samples where the source is identified by the taxonomy term "rodentia", then we want the system also to return information about these samples. To help automate this, we can define a derived function that takes a name from the taxonomy as its argument and returns all names below that given name. This function has a recursive definition:

```
define acceptable_alternative_names(s in string) ->> string
  (name(t1 in taxonomy_entry such that name(is_of_taxa(t1))=s)
   union
   acceptable_alternative_names(
     name(t2 in taxonomy_entry such that name(is_of_taxa(t2))=s)));
```

Two pieces of extra information are recorded in the database system's metadata: (i) information with the attribute (in this case `organism_ncbi`) indicating that values for this attribute are taken from a hierarchical vocabulary, and (ii) the name of the function that can derive alternative acceptable values for this attribute by traversing the hierarchy recursively. When a query is submitted to the database management system, the system checks whether a value has been specified for such an attribute and, if it has, that part of the query is replaced by code that tests values actually stored in the database with the string specified by the user, and also with alternative terms below the given term in the hierarchical vocabulary.

As a second example, we have stored terms and relationships from the Gene Ontology[3] (GO) in the P/FDM database. The schema used for this is:

```
declare go_term              ->> entity
  declare name(go_term)       -> string
  declare term_type(go_term)  -> string
  declare isa(go_term)       ->> go_term
  declare partof(go_term)    ->> go_term
  key_of go_term is name, term_type;
```

The functions `isa` and `partof` are multi-valued (indicated by the double-headed arrows), as are the inverse relationships (`isa_inv` and `partof_inv`) that are maintained automatically by the P/FDM system. The function `isa_inv` can be applied recursively to find all more specialised terms of a given term:

```
define all_more_specialised_terms(t in go_term) ->> go_term
  (isa_inv(t) union all_more_specialised_terms(isa_inv(t)));
```

A function that identifies all terms that are subparts of a given GO term in accordance with the logical relationships in the GO Usage Guide[4] can also be defined, but care must be taken in interpreting the results of this function since "partof" is a weak relationship in GO in the sense that it means "can be a part of", not "always a part of".

[3] http://www.geneontology.org/

[4] http://www.geneontology.org/doc/GO.usage.html