Use of OWL within the Gene Ontology

Christopher J Mungall*[1] Heiko Dietze[1] David Osumi-Sutherland[2]

Abstract. COPY FROM GDOC

1 body

THIS IS CUT AND PASTED FROM GDOC WHICH STILL REMAINS PRIMARY SOURCE

The Gene Ontology (GO) is one of the most successful and widely used ontologies in the life sciences and indeed in the history of knowledge representation. Commonly conceived of as a simple direct acyclic graph, the GO is actually well-axiomatized in OWL and makes use of a lage stack of OWL tools. Here we outline some of the lesser known features of the GO, describe the GO development process, and our prognosis for future development in terms of the OWL representation.

Introduction

The Gene Ontology (GO) is a bioinformatics resource for describing the roles genes play in the life of an organism, covering a variety of species from humans to bacteria and viruses. The GO is commonly presented as a Directed Acyclic Graph (DAG) G = ¡V,E¿, where each vertex in V is a particular gene descriptor, and E is a set of labeled edges connecting two vertices in V. The GO is designed to be used in combination with a database D = ¡A, M¿ where M is a set of molecular entities (e.g. genes or the products of genes) and A is a set of associations where each association connects an element of V with an element of M. Currently GO has some 40k vertices, nnn edges, and the combined set of databases using GO have xxx associations covering yyy genes in zzz different species[**?**].

The GO is used in a number of ways. The simplest way is to interrogate a database, for example to find the set of descriptors for a gene, or to find the set of genes for a descriptor (making use of E). One of the most common uses is to find a functional interpretation of a set of genes, a so-called enrichment test, for example given a a set of genes that are switched on as a result of an organism being exposed to an environmental toxin[REF]. Another use is as a component of a diagnostic tool for finding causative genes in rare diseases[REF]. As of today, the GO has been cited NNN times (and this is an under-representation of the true use), with XXX tools dedicated to using the GO, integrated into MMM databases.

This simple formulation of the GO is popular, but somewhat outdated, as the GO has incorporated an ever increasing number of OWL constructs over the years. The core ontology graph G maps to SubClassOf axioms – either between two named classes (so called is_a links) or between a class an an expression of the form R some Y. Behind the scenes there are additional axioms, some of which are described here.

The Axiomatic Structure and Logic of the GO

The GO consists of 40k classes, but also includes an import chain that brings in an additional nnn classes from 5? additional ontologies. The majority of the axioms in this import chain are within the EL++ profile, allowing for the use of faster reasoners (axioms outset this subset are described later).

The breakdown of axiom types and expression types is as follows [TABLE]. As can be seen, the most common expression types in GO are existential restrictions and intersections, with the latter used almost entirely within equivalence axioms.

The entire ontology reasons in nn s in Elk, and nn minutes in Hermit.

Equivalence axioms in GO

The meat and potatoes for reasoning in GO are the equivalence axioms, most typically of a genus-differentia form, i.e. X EquivalentTo G and R1 some Y1 and Rn some Yn. These are known colloquially in GO as cross products, since the set of such defined classes X can be drawn from the cross-product of the set G and the sets Y1..Yn.

The existence of these axioms allow us to use reasoners to automatically classify the GO, something that is vitally important in an ontology with so many classes. This was previously done entirely by hand, which was time-consuming and error-prone. We aim to follow the Rector Normalization pattern[REF] as closely as possible but in practice this is hard, as we are still in the midst of refactoring a largely hardcoded structure.

These axioms can be broken down into intra-ontology and inter-ontology equivalence axioms. The inter-ontology axioms have the added benefit of connecting different ontologies used for classification of different types of data.

We use a number of different Object Properties in these axioms, largely taken from the OBO Relations Ontology [link to RO website].

Constraints in the GO

As well as automatic classification, there is a need for automated quality control processes. The GO has a large QC pipeline, a subset of which is implemented via standard OWL reasoning operating over the axioms in GO.

The majority of constraints in GO are encoded via disjointness axioms; many of the relations used in GO are quite general and thus domain and range constraints are less useful. We achieve a more powerful conextual domain-range type assertions using disjointness axioms. For example, the part_of relation is flexible regarding whether is it used between processes (for example, a GO biological process such as flower development) or between static material entities (for example, a GO subcellular component, such as synapse). This generality limits the utility of domain and range. However, the RO includes axioms of the form: part_of process DisjointWith part_of some continuant Which prohibits category-crossing uses of part_of which would be invalid.

Disjointness axioms are also used in the traditional way, between siblings in a taxonomic classification.

We frequently have need to encode spatial and temporospatial constraints. For example, most of the cells in a complex organism such as yourself consist of a number of compartments, including the nucleus (the central HQ, where most of your genes live) and the cytosol (a kind of soup full of molecular machines doing their business). Its not enough to simply state that the cell and cytosol are disjoint classes. We also want to encode what in RCC8 terms is the disconnected relation, the fact that no parts are shared in common (made impossible by the existence of a membrane barrier between the two). We do this using General Class Inclusion axioms (GCI axioms), e.g. (part_of cytosol) DisjointWith (part_of nucleus)

Constraints are also used to check the contents of databases. For example..

Another common type of contsraint in the GO are so-called taxon constraints [REF Deegan] TODO

Smuggling OWL expressions into Databases

[REF: increased expressivity of GO annotations]

Logic is not the only fruit: Annotations in the GO

Say something about how its not just logic thats important to us in GO. Highlight axiom annotations. Mention the terminological confusion over annotation.

The GO Development environment

Protege and OBO-Edit dual editing Protege plugins TermGenie OWLTools and Oort Continuous Integration VCSs and continued use of OBO

Discussion of obstacles, what we need, when do we want it. Inferring subclass of part_of some Y. More OE like experience in Protege. More tooling. Something like Maven for ontologies, especially with dependency/version management.

Mention that GO development is closely coordinated with others like CL, UBERON, both in terms of ontology integration and shared infrastructure

Limitations of MIREOT. We have to mention OBO Foundry efforts here even though in many ways its ceased to be of relevance...

Downstream use of the GO. More OWLishness required!

The detailed OWL axiomatization of GO is primarily used within the GO consortium, as a tool for automation and QC during the ontology development lifecycle, and as a database QC tool. However, these more advanced axioms are still not widely used by the many groups developing software the embed or leverage the GO.

Discussion on why this is and whats to be done. better integration of statistical/probablistic views and logical.

What does the future hold?

GO was designed to be used with a simple database structure, pairwise associations between nodes in the GO graph and molecular entities.

Where were going now: describing complex interactions. How should this be done? ABox or TBox or some mix?

FIGURE: Noctua

Conclusions

OWL is great but we want better support, easier tools, .

2 Conclusions

YADDA