Use of OWL within the Gene Ontology

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

Abstract. 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 terminology structured as a directed acyclic graph, the GO is actually well-axiomatized in OWL and makes use of a large 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.

1 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 way the GO is most commonly presented in publications elides much of the underlying axiomatization and formal semantics. The most common conception is 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 not of much use to biologists in itself - the value comes in how databases use the GO to "annotate" genes and molecular entities. A database D is a pair $< A, M >$ 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, 100k edges, and the combined set of databases using GO have xxx associations covering yyy genes in zzz different species[2].

The users of the GO apply it 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 the edges in 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 set of genes that become active 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[**?**]. 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This simple view of the GO is popular, but outdated, as the GO has incorporated an ever increasing number of OWL constructs over the years.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 expression of the form R *some* Y. Behind the scenes there are additional axioms, some of which we will describe in this manuscript.

2 The Axiomatic Structure of the GO

The GO consists of over 40,000 classes, but also includes an import chain that brings in an additional nnn classes from 6 additional ontologies. The majority of the axioms in this import chain are within the EL++ profile, allowing for the use of faster reasoners (we will describe the small number of axioms outset this profile further on).

For release purposes, the GO is available as a limited "standard edition" which exlcudes imports and external ontologies and a complete edition called "go-plus"[1]. There are additional experimental extensions which are not discussed here (but we encourage reasoner developers to contact us for access to these for testing purposes).

Table 1 shows the breakdown of axiom types and expression types. As is evident, existential restrictions and intersections are frequently used, with the latter used entirely within equivalence axioms.

Construct	**Usage count**
Axiom	556088
EquivalentClasses	27108
IntersectionOf	27108
SubClassOf	106063
AnnotationAssertion	370394
DisjointClasses	127
SomeValuesFrom	36125

Table 1. Axiom or expression types used in the GO

The part of GO that is most typically exposed to users are the SubClassOf axioms (together with annotation assertions), which is weak in terms of expressivity but delivers the qery abilities required by most users.

The entire ontology reasons in seconds in Elk[7], and 10 minutes in Hermit (on a standard laptop or workstation).

2.1 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* R_1 *some* Y_1 *and* R_n *some* Y_n. These were historically referred to in GO as 'cross products' [9], since the set of such defined classes X are a subset of the cross-product of the set G and the sets Y.

The existence of these axioms allow us to use reasoners to automatically classify the GO, something that is vitally important in an ontology with such a large number of classes. In the current release version of GO, over 32,000 SubClassOf

[1] http://geneontology.org/page/download-ontology

axioms were inferred by reasoning, representing a substantial efficiency gain for ontology developers.

2.2 Inter-ontology axioms

The subset of GO most commonly exposed comprises the so-called "intra-ontology" axioms, but GO also contains a rich set of inter-ontology axioms, leveraging external ontologies (The plant ontology, Uberon, the Cell type ontology, and the CHEBI ontology of chemical entities).

The primary use case for the inter-ontology axioms is to allow for modularized development and to automatically infer the GO hierarchy. Additionally, inter-ontology axioms have the added benefit of connecting different ontologies used for classification of different types of data.

The inter-ontology axioms are present in the go-plus edition of GO, but not the core version. To avoid importing large external ontologies in their entirety, we build "import modules" using the OWL API Syntactic Locality Extractor Module.

2.3 Relations in the GO

We use a number of different Object Properties in these axioms, taken from the OBO Relations Ontology[2]. We rely heavily on Transitivity, *SubPropertyOf* and *SubPropertyChain* axioms.

The majority of relations in RO have an Inverse declared. Typically within the GO we only use one form, and we never use the *Inverse* construct in a property expression. One exception to this is the use of *partOf* and *hasPart*. We frequently create axioms with an existential restriction using partOf (this forms the core partonomy of the GO), but have more recently started creating hasPart axioms[1]. This takes us outside the profile covered by Elk. However, we are not reliant on the inverse axiom connecting these two properties for classification purposes. We have on occasion used this inverse axiom in combination with HermiT to successfully detect unsatisfiable classes.

2.4 Constraints in the GO

As well as automatic classification, we also make extensive use of reasoning as part of our quality control pipeline, both for ontology validation, and for the validation of data about genes coming from external databases.

We encode the majority of constraints in GO as disjointness axioms. Domain and range constraints on object properties play less of a part. We achieve more powerful contextual domain-range type assertions using disjointness axioms. For example, the 'part of' relation is flexible regarding whether is it used between two processes (such as those found in the GO 'biological process' branch) or between material entities (for example, a GO subcellular component, such as synapse).

[2] http://code.google.com/p/obo-relations

This generality limits the utility of domain and range. However, the RO includes axioms of the form: 'part of' process DisjointWith 'part of' 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, although these are typically underspecified.

We frequently have need to encode spatial and spatiotemporal 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). It is not enough to simply state that the cell and cytosol are disjoint classes. We also want to encode what in RCC8 termiology is the "disconnected" relation, the fact that there are no shared parts (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)
```

In some cases we can structurally simplify the axiom by using named classes such as 'cytosolic part' and 'nuclear part'.

Another common type of contsraint in the GO are so-called 'taxon constraints' [4]. The basic idea here is that the GO covers biology for all domains of life, from single-celled organisms to humans. However, many of the classes are applicable to specific lineages. For example, in describing the function of genes in a poriferan (sponge), it would be a mistake to use the GO class brain development, or any of its descendant classes, as these simple organisms lack a nervous system of any type. Whilst we would hope a human curator would not make such an error, the same cannot be said for algorithmic prediction methods that make use of the 'ortholog conjecture'[13] to infer the function of a gene in one species based on the function of the equivalent gene in another species. Sponges have many of the same genes found in other animals that form synapses in the nervous system (the jury is out on whether this is a case of evolutionary loss or a case of co-option). Here it is useful to have a knowledge-based approach to validation of computational predictions.

The most obvious way to encode taxon constraints is using universal restrictions and complementation expressions; however, this has the disadvantage of being outside EL++. We instead encode taxon constraints as disjointness axioms[8].

One place where we use UnionOf constructs is in the GO-specific extensions of the taxonomy ontology where we create grouping classes. For example, the grouping "Prokaryota" would not be found in the taxonomy ontology as it constitutes a paraphyletic group - nevertheless it is useful to refer to refer to these groups, so we create these as union classes (in this case, equivalent to the union of "Eubacteria" and "Archaea"). The increase in expressivity beyond EL++ is not a practical issue here as the groupings do not change frequently so we prereason and assert the direct subclass inferences (here between Prokaryotes and the class for cellular organisms).

2.5 Annotation axioms

In addition to the logic axioms described above, GO makes heavy use of annotation assertion axioms, as the textual component of GO is important to our users. In particular, textual definitions, comments and synonyms (in addition to labels) are the annotation properties we use most commonly.

One of main factors that allowed us to move to OWL was the introduction of axiom annotations. We attempt to track provenance on a per-axiom level, so this feature is vital to us.

3 The GO Development environment

3.1 Transitioning to OWL

The GO was not born as a Description Logic ontology. In order to be able to take advantage of automated reasoning, it was necessary for us to *retrospectively* go back and assign equivalence axioms and other OWL axioms to existing classes, some of which date back to the inception of the project. This is in contrast to ontologies "born" as OWL following the Rector Normalization pattern[12] in which classes are *prospectively* axiomatized, at the time of creation.

The process of restrospective axiomatization was assisted in part by the detailed design pattern documentation maintained by the GO editors – see for example the documentation on developmental processes[3]. This made it possible to use lexical patterns to derive equivalence axioms[9]. For example, if a class C has a label "X differentiation" then we derived an axiom *C EquivalentTo 'cell differentiation' and results_in_acquisition_of_features_of some X*. X is assumed to come from the OBO cell type ontology, and if no such X exists then we add this.

However, the axiomatization process frequently revealed cryptic inconsistencies and incoherencies, both within the GO, and between the GO and other ontologies. Some of these are trivial to resolve, whereas others require a massive conceptual alignment of two domains of knowledge. One such case was the alignment of a biology-oriented view of metabolic processes with a chemisttry-oriented view[5].

3.2 Editing tools

The GO development environment is a hybrid of different tools and technologies.

In the past, the GO developers exclusively used OBO-Edit (OE)[3] for construction and maintenance of the ontology. OBO-Edit only supports a subset of OBO-Format, which corresponds roughly to EL++, with the addition of other restrictions, such as limited ability to nest class expressions. However, the main limitation of OBO-Edit is the lack of integration with OWL Reasoners such as Elk.

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

Protege represents a superior environment for logic-based ontology development, but unfortunately lacks much of the functionality that makes OBO-Edit a productive and intuitive tool for the GO developers. These features include powerful search and rendering, visualization, inclusion of existential restrictions in hierarchical browsing, and annotation editing customized for our annotation property vocabulary.

To overcome this we have been moving to a hybrid editing environment, whereby developers use a mixture of OBO-Edit and Protege. The source ontology remains in OBO-Format, with the developers using OWLTools[**?**] to perform the conversion to OWL and back. Developers are careful to remain within the OBO subset of OWL.

At first we employed this hybrid strategy tentatively, with the developers using Protege primarily as a debugging tool (for example, explanation of inferences leading to unsatisfiable classes). However, developers are gradually embracing Protege for other parts of the ontology development cycle, such as full-blown editing.

To facilitate this transition, we have been working with other software developers to create plugins that emulate certain aspects of the OE experience. These include an OBO-Edit style annotation viewer and editor[4], a plugin that manages the obsoletion of classes according to GO lifecycle policy[5], and a partial port of the OE graph viewer[6]. **??**.

3.3 Web based templated term submission

Biological data curators frequently need new classes for describing the genes they are annotating. Often these classes fall into particular compositional patterns with placement in the subsumption hierarchy calculated automatically. In the past the sole method for data curators to obtain new classes was through a sourceforge issue tracking system, leading to bottlenecks.

To address this we created TermGenie[**?**], a web-based class submission system that allows curators to generate new classes instantaneously, provided they pass a suite of logical, lexical and structural checks. TermGenie submission can be according to either pre-specified templates, or "free-form" submissions.

Currently we specify the templates procedurally as javascript code, and we are currently exploring the use of Tawny-OWL[**?**] as the templating engine.

3.4 Smuggling OWL expressions into Databases

In order to avoid overloading the ontology with two many named classes, we have created an "annotation extension" system whereby data curators can composed their own class expressions for describing genes[6]. The expressivity of the system

[4] https://github.com/hdietze/protege-obo-plugins
[5] https://github.com/balhoff/obo-actions/downloads
[6] https://code.google.com/p/obographview/

is deliberately limited to refining a base class using one or more existential restrictions.

This system has so far appeared to be a useful balance between expressivity and simplicity. One problem is that data curation takes place outside an OWL environment, so any logical errors (for example, violation of a domain or range constraint) are not caught until the curators submit their data to the central GO database, where we perform reasoner-based validation.

3.5 Ontology build pipeline

As the GO evolved from being a single standalone artefact to modular entity with a number of derived products we constructed an ontology verification and publishing pipeline.

As is common in the bioinformatics world, we specify and execute our pipeline using UNIX Makefiles, which allows the chaining together of dependent tasks that consume and produce files.

We developed a command line utility that acts as a kind OWL swiss-army knife, with the original name of OWLTools[**?**]. We developed OWLTools according to the unix philosophy, with a view to integration with Makefile-type pipelines. It is primarily a simple wrapper onto the OWL API, and allows the execution of tasks such as checking if an ontology is incoherent, generating subsets of ontologies using the OWL Syntactic Locality Module Extractor, and so on.

This pipeline is executed within a Continuous Integration framework[10].

4 Challenges and obstacles

4.1 Challenges of working with multiple ontologies

We aim to follow the Rector Normalization patter, avoiding manual assertion of polyhierarchies, instead leveraging modular hierarchies. Often these hierarchies fall in the domain of an ontology external to GO, which presents a number of challenges. This was one of the original motivations for the creation of the Open Biological Ontologies (OBO) library, to lower the barrier for interoperation, by ensuring all federated ontologies were open, orthogonal and responsive to any requirements for improvement or change. Even with these barriers lowered, challenges remain. Ontologies developed by different groups often reflect different perspectives, design patterns and hidden assumptions that can be hard to reconcile. The initial axiomatization of one ontology using another often reveals multiple unsatisfiable classes and invalid inferences. This can be time-consuming to repair. The key here is early, prospective integration, rather than after-the-fact.

There is also a deficit of tooling to support working in a multi-ontology environment. Naive construction of import chains results in highly inefficient transfer of large RDF/XML files over the web. The resulting infrastructure is fragile, with

multiple points of failure. Versioning becomes of paramount importance, because simple changes in an imported ontology can wreak havoc, causing mass unsatisfiability, or loss of crucial inferences. Multiple partial solutions exist, but none are perfect. BioPortal provides URLs for individual versions of ontologies, but require an API key, which does not work well with owl imports.

The parallels with software development are obvious. As ontologies such as GO move from being monolithic to modular, we need the equivalent of dependency management and build tools such as Maven, and we welcome efforts such as the revent OntoMaven project[11].

Biological ontologies do not always modularize as cleanly as software libraries. For example, there are multiple mutual dependencies between the cell type ontology and GO (the former relies on the latter to describe what cells have evolved to do, the latter relies on the former to describe the development of these cells). This presents additional challenges.

4.2 Getting OWL into the mainstream

TODO: 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.

Currently we have a requirement that ontology developers can switch between use of obo-edit and Protege. Unfortunately, the reasoning support in obo-edit is poor (not integrated with Elk or any standard OWL reasoner). This means that there is effectively no inferred class view, so we cannot rely on dynamic classification if developers wish to see a rich and complete hierarchy. As a workaround, we have a batch process that mass asserts all direct inferred subclass axioms (and tags these as being asserted via an axiom annotation). In theory these can be deleted and recreated en-masse. In practice this is frankly a rube goldbergesque system that has evolved ad-hoc over the years, and we are planning to overhaul this.

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.

5 What does the future hold?

https://github.com/phenoscape/owlet

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

6 Conclusions

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

A small amount of constructs go a long way: EquivalentClasses, SomeValuesFrom, DisjointWith, SubPropertyChainOf Elk was a game-changer In many aspects GO axiomatization is of a similar expressivity to earlier DL ontologies like GRAIL. Lessons? Takes a long time for tools to harden, specs to mature and technology to percolate Foundations of OWL2 are solid (specifications, core APIs) but we need more tools that can be used as part of the ontology development lifecycle. Examples are things like owlet. How can we get OWL axioms to be used more downstream of the ontology development lifecycle? Should they be? Is the future in integration between DLs and the kinds of statistics and probabilistic reasoning more common in bioinformatics

7 Conclusions

YADDA

References

1. T.Z. Berardini, V.K. Khodiyar, R.C. Lovering, and P. Talmud. The Gene Ontology in 2010: extensions and refinements. *Nucleic Acids Research*, 38(Database Issue):D331–D335, 2010.
2. JA Blake, M Dolan, H Drabkin, DP Hill, L Ni, D Sitnikov, S Bridges, S Burgess, T Buza, F McCarthy, D Peddinti, L Pillai, S Carbon, H Dietze, A Ireland, SE Lewis, C.J. Mungall, et al. Gene ontology annotations and resources. *Nucleic Acids Research*, 41(D1):D530–D535, 2013.
3. John Day-Richter, Midori A Harris, Melissa Haendel, Gene Ontology OBO-Edit Working Group, and Suzanna Lewis. OBO-Edit–an ontology editor for biologists. *Bioinformatics*, 23(16):2198–2200, August 2007.
4. J Deegan, E Dimmer, and C. J. Mungall. Formalization of taxon-based constraints to detect inconsistencies in annotation and ontology development. *BMC bioinformatics*, 11(1):530, 2010.
5. David P Hill, Nico Adams, Mike Bada, Colin Batchelor, Tanya Z Berardini, Heiko Dietze, Harold J Drabkin, Marcus Ennis, Rebecca E Foulger, Midori A Harris, Janna Hastings, Namrata S Kale, Paula de Matos, Christopher Mungall, Gareth

Owen, Paola Roncaglia, Christoph Steinbeck, Steve Turner, and Jane Lomax. Dovetailing biology and chemistry: integrating the Gene Ontology with the ChEBI chemical ontology. *BMC genomics*, 14(1):513, January 2013.

6. Rachael P Huntley, Midori A Harris, Yasmin Alam-Faruque, Judith A Blake, Seth Carbon, Heiko Dietze, Emily C Dimmer, Rebecca E Foulger, David P Hill, Varsha K Khodiyar, Antonia Lock, Jane Lomax, Ruth C Lovering, Prudence Mutowo-Meullenet, Tony Sawford, Kimberly Van Auken, Valerie Wood, and Chris Mungall. A method for increasing expressivity of Gene Ontology annotations using a compositional approach. *BMC Bioinformatics*, 15(1):155, 2014.
7. Yevgeny Kazakov, Markus Krötzsch, and František Simančík. Elk reasoner: Architecture and evaluation. *CEUR Workshop Proceedings*, 858, 2012.
8. Chris Mungall. Taxon constraints in owl. http://douroucouli.wordpress.com/2012/04/24/taxon-constraints-in-owl/, 2012.
9. Chris Mungall, Michael Bada, Tanya Z Berardini, Jennifer Deegan, Amelia Ireland, Midori A Harris, David P Hill, and Jane Lomax. Cross-product extensions of the Gene Ontology. *Journal of Biomedical Informatics*, 44(1):80–86, 2011.
10. Chris Mungall, Heiko Dietze, Seth J Carbon, Amelia Ireland, Sebastian Bauer, and Suzanna Lewis. Continuous Integration of Open Biological Ontology Libraries. pages 1–4, 2012.
11. Adrian Paschke. Ontomaven: Maven-based ontology development and management of distributed ontology repositories. *arXiv preprint arXiv:1309.7341*, 2013.
12. Alan L. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including OWL, 2003.
13. Paul D Thomas, Valerie Wood, Chris Mungall, Suzanna E Lewis, and Judith A Blake. On the Use of Gene Ontology Annotations to Assess Functional Similarity among Orthologs and Paralogs: A Short Report. *PLoS computational biology*, 8(2):e1002386, February 2012.