Reconciling the Semantics of DAG and OWL Ontology Representations

Robert Stevens, Sean Bechhofer, Ulrike Sattler, Phillip Lord
Department of Computer Science
University of Manchester
Oxford Road
Manchester
M13 9PL UK
robert.stevens@cs.man.ac.uk

August 22, 2005

Abstract

The bio-ontologies community falls into two camps: First we have biology domain experts, who actually hold the knowledge we wish to capture in ontologies; second, we have ontology specialists, who hold knowledge about techniques and best practice on ontology development. In the bio-ontology domain, these two camps have often come into conflict, especially where pragmatism comes into conflict with perceived best practice. One of these areas is the insistence of computer scientists on a firm semantic basis for the representation language being used. In this article, we will first describe why this community is so insistent. Second, we will examine the semantics of the Web Ontology Language (OWL) and the directed acyclic graph (DAG) used by the Gene Ontology. Finally we will reconcile the two representations. The ability to exchange between the two representations means that we can capitalise on the features of both languages.

1 Introduction

In this paper, we investigate the reconciliation of the representations used for the Gene Ontology (GO) [3] and that used for the ontologies represented in the W3C standard OWL (Web Ontology Language)[1]. Different knowledge representation languages make statements about the knowledge they capture in different ways. The semantics of these languages tell both humans and computers how to interpret statements made in those languages. Different languages have varying expressivity and computational support; consequently there is often a need to exchange between languages to take advantage of their characteristics. In order to do so, we have to make sure that the knowledge captured in statements in one language is not changed when a transformation is made into statements in another language. Hence, the semantics of one language need to be reconciled with the semantics of the other.

[1] http://www.w3.org/TR/owl-guide/

The GO has become the *de facto* standard for describing the principle attributes (the molecular function, biological process, and cellular component) of gene products across many databases [3, 4]. It succeeds in the major aim of an ontology in providing a common, shared understanding of the concepts used to describe those attributes. It does this by providing terms used to label those concepts as well as textual definitions of those terms.

The GO and related ontologies in the Open Bio-Ontologies[2] (OBO) project often use an ontology representation language developed in-house – the Directed Acyclic Graph (DAG) [4]. It has the tremendous advantage of simplicity and has enabled the Gene Ontology Consortium (GOC) to develop to its current status [2].

The OBO site states that submitted ontologies can be presented in the DAG format or in OWL. Being a collection of bio-ontologies, it would be useful to be able to exchange ontologies between the two formats. Indeed, this has already been attempted in the Gene Ontology Next Generation (GONG) project [9]. In this project, the predecessor of OWL, DAML+OIL, was used to create definitions of GO classes by decomposing the phrases used for the terms. These descriptions were then submitted to a description logic (DL) reasoner to infer a subsumption hierarchy. Using this technique, it was possible to detect defects in the subsumption hierarchy of a region of GO. Once a new classification had been inferred, the aim would be to translate back from DAML+OIL to GO's original DAG representation. In the GONG project, we made this translation, but made simplifying assumptions. In this article, we take a closer look at how such a translation could be accomplished.

In Section 2 we explain why computer scientists, in particular, like to have a strong semantics in their representation languages. In Sections 3 and 4 we outline the semantics of GO's representation and that of OWL. Finally, in Section 5, we attempt to reconcile the two representations.

2 Why Do Computer Scientists Care So Much About Semantics?

The knowledge representation community within computer science has the aim of representing knowledge in a form both understandable by humans and a computationally amenable. Computers, of course, do not have the same facility to "understand" knowledge captured in an ontology as do the human users of that ontology. To a computer, the label on a concept (the term) is not comprehensible. Taking the example in Figure 1, a human will understand that "an instance of person cannot be both a Man and a Woman at the same time"[3]; the computer will not.

The need to capture knowledge with high-fidelity and interpret it unambiguously is enabled by having a representation language with strongly defined semantics. In the same way that a C programming language compiler must *understand* what each of the language components means in terms of constructing a programme that runs on a particular machine, so must a computer understand what each of the expressions in the description of some knowledge represents. The computer's *understanding* is encapsulated in the semantics of the language – be it a programming language or knowledge

[2] `http://obo.sourceforge.net`

[3] at least not in this view of the world!

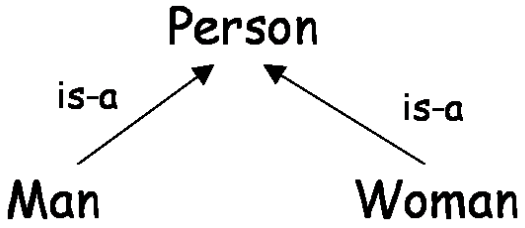

Figure 1: A simple ontology of `Person`.

representation language.

Figure 1 shows a simple ontology of `Person`, with two child classes of `Man` and `Woman`. As human users we understand, or believe we understand, what is being represented in such an ontology: "there are two kinds of Person, namely Man and Woman". We can, however, ask several supplementary questions of this ontology:

- Are all instances of `Man` also instances of `Person`?
- Are `Man` and `Woman` the only kinds of `Person` that exist?
- Is it possible for an instance of `Person` to be both a `Man` and a `Woman`?

We might also extend our description of `Person` to say that they contain `Gonad`. So, for `Man`, we might say `Man has-part Testis`. Again, we might ask ourselves several additional questions:

- How many `Testis` does a man have?
- Can a man only have `Testis` or may he have other parts?
- Does having a `Testis` make an instance of `Person` a man?
- Are `Testis` the only gonads a man can have?
- Do all `Man` have `Testis`?
- Are all `Testis` parts of `Man`?
- May I say anything more about the parts a `Man` has?

As human users of the ontology shown in Figure 1 we may understand, deduce or infer the answers to these questions, or we may not; it is certain, however, that the computer will not do so. It is in the semantics of the representation language that the answers to such questions can be couched. It is part of the semantics of a language that says whether two children of a concept are overlapping, that is, is it possible to be both a `man` and a `Woman`. For a computer to know that the only possible kinds of `Person` are `Man` and `Woman` the fact has to be explicitly stated.

Returning to human users of an ontology, the semantics that make it possible for a computer to interpret the representation, also make it possible for a human to do so. A user might believe they understand what is represented in the ontology shown in

Figure 1, but dangerous assumptions might be made when doing so. If the representation language has a precise semantics, then what is expressed in that language can also be captured with high-fidelity. So, while precision is vital for computers, it is also extremely helpful for humans as they can be explicit and precise about the knowledge they are attempting to capture.

3 OWL Semantics

OWL-DL[6] is an ontology language based on description logics (DLs), which are a family of logic-based knowledge representation formalisms describing *objects*, *classes* and the *relationships* between them [1]. Most DLs are fragments of standard first order logic. Originally, they were designed to give a unified logical basis to various well-known traditions of knowledge representation: frame-based systems, semantic networks, and have found various applications in conceptual modelling and as a logical underpinning of ontology languages [1]. OWL-DL is based on an expressive DL, *i.e.*, it provides a wealth of constructors to describe complex class expressions from atomic classes and relationships. In this section, we will only use a small portion of OWL-DL's expressiveness to highlight its core features.

The semantics of OWL are best understood when talking about *objects* that are *instances* of *classes*, and that are related to other objects via *relations*.

An object can be *an instance* of a class, and a class can be a sub-class of another class. For example, the object Robert is an instance of the class `Man` which, in turn, is a sub-class of `Person`. The meaning of the sub-class relationship is that all instances of the sub-class, `Man`, are also instances of its super class(es), `Person`. In OWL-DL, to describe a class, we can describe it in terms of other classes (*e.g.*, saying that `Man` are "`Person` *and not* `Woman`") and of properties of its instances.

In Section 2 we described an ontology with two classes, `Man` and `Woman`, which are both sub-classes of `Person`. In OWL-DL, we can explicitly declare that these two classes are *disjoint*; that is, it is not possible for an object to be an instance of both classes. Similarly, we have to decide whether it is possible for an instance of `Person` to be neither an instance of `Woman` nor of `Man`. Again, if this can not be the case, we can declare `Person` as *covering* `Woman` and `Man`.

class (`Man` complete `Person`
restriction (`has-part` someValuesFrom `Testis`))
Paraphrase: Men are Persons that have, *amongst other things*, *some testis*.
class (`Woman` complete `Person`
restriction (`has-part` allValuesFrom complementOf(`Testis`)))
Paraphrase: Women are Persons that have, *amongst other things*, only *no testis*.

Figure 2: Description and paraphrase of `Man` and `Woman`.

As stated above, OWL-DL also allows us to describe a class by describing how its instances are related to other objects. For example, the first restriction in Figure 2 states that an instance of `Man` has an instance of `Testis` related to it via the relation `has-part` property. Thus, if we know that `Robert` is a `Man`, we also know that he has a part that is a testis–however, he might have several of those and also other parts,

as well. As this statement only says something about the existence of a relationship to another object, it is called an *existential* restriction. Additionally, OWL-DL also allows to make *universal* restrictions: *e.g.*, we can say that an instance of `Man` is related via the relation `has-part` *only* to instances of `Testis`. As a consequence of such a statement and `Robert` being an instance of `Man`, if we find an object related to `Robert` via `has-part`, then this object has to be a `Testis`. Moreover, we can combine these statements and also use more complex class expressions. In Figure 2, we say that all parts a `Woman` has are *not* `Testis`. The semantics of OWL-DL mean that we are saying all instances of `Man` have a part that is an instance of `Testis`. We are not, however, saying that all instances of `Testis` are parts of a `Man`.

In all of these examples we have only stated restrictions concerning `Man`, `Woman` and the objects to which they are related by the `has-part` relation. We have not restricted any other relationships we might choose to describe, such as *has-mother*.

OWL-DL also enables us to distinguish between "complete" and "partial" class definitions: that the definition of `Man` is complete means that any `Person` with a `Testis` is recognised to be a `Man`. In contrast, a partial definition only represents necessary conditions, but not sufficient ones: for example, as shown in Figure 3, we can define the class `Eunuch` as a subclass of `Person`; so every `Eunuch` has parts none of which are `Testis`, but not everyone with no `Testis` is a Eunuch.

class (`Eunuch` partial `Person`
complementOf (restriction (`has-part` someValueFrom `Testis`)))
Paraphrase: Eunuchs are Men that have, *amongst other things*, *no testis*.

Figure 3: Description and paraphrase of Eunuch

Due to its description logic underpinning, OWL-DL ontologies can be submitted to a DL reasoner which provides useful reasoning services. Most importantly, a reasoner can decide the *consistency* of each class defined in the ontology and it can compute the implicit class hierarchy. For example, given the statements made so far, the reasoner infers that a `Eunuch` is, in fact, a subclass of `Woman`. This seems a little counter-intuitive, so we might also assert that a `Eunuch` is a subclass of `Man`. The reasoner will then tell us that Eunuch is *inconsistent*: there can be no instances of it. In this case, its probably our definition of `Man` that is a poor model of reality. The inconsistency of the `Eunuch` forces us to re-examine this model. The precise and explicit nature of models in OWL-DL allow us to check the knowledge we have captured and enable them to be interpreted correctly.

For a complete description of OWL-DL, we refer the reader elsewhere [6]. Here, we have only used a small part of OWL-DL's expressiveness. OWL-DL also enables us to use relations in both directions (*e.g.*, we can also define gonads in terms of the organisms they occur in), state that a relation such as has-part is transitive (*e.g.*, making a `SemiNiferousTubule` part of a `Testis` also makes it part of a `Man`), and restrict the numbers of objects to which an instance is related by a specific relationship(*e.g.*, restricting the numbers of gonads a `Person` has to 2). It should be enough, however, to indicate that the formal semantics of OWL enables both the author and a computer to "understand" precisely what has been stated, and enable software such as a reasoner, to deduce implicit knowledge from such representations [7, 8].

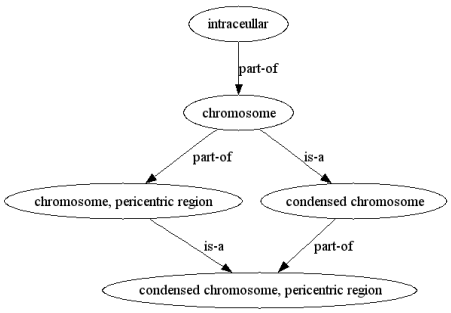

Figure 4: A GO DAG with both `is-a` and `part-of` relationships.

4 The Semantics of the Gene Ontology's Representation

The aim of this section is to elucidate the semantics of GO's encoding and not to examine the biology captured in that encoding. There is need, however, to sometimes look at the biology in order to understand the encoding.

The GO is encoded as a Directed Acyclic Graph (DAG) (see Figure 4). A DAG is a form of multi-hierarchy of nodes and arcs. The nodes form the terms or classes in the ontology and the arcs the relationships. Any one class can have many parents and many children. More than one kind of relationship can be used. The *directed* nature of the graph comes from the directional nature of the relationships: `A` is a parent of `B`, not vice versa.

Terms label nodes, which may be interpreted as a class (all the instances of `Man` or `Woman`). The term refers to a node and all of its children [5]. GO also allows synonyms to be recorded; alternative labels for nodes or classes. In GO's DAG representation, there is no way of stating whether classes (*e.g.*, `Man` and `Woman`) are overlapping or disjoint [5].

Sufficiency conditions are not mentioned within GO. For GO, definitions are natural language, free text descriptions that 'define' what a *term* means. These are used by annotators and GO curators alike when using GO.

GO uses two types of relationships. The '`is-a`' relationship points from a child (more specialised) to a parent (more generalised) term [5]. Figure 4 shows an example of a GO DAG with both kinds of relationship. The `part-of` relationship, in GO's usage, talks about parts and parents, not *parts* and *wholes*, as is conventional. In this way, we can see what some [10, 9] have called *orphan* nodes (see Figure 4), where the node is `part-of` another node, but is not a kind of any node. Conventionally, this would be a child with no parent, *i.e.*, an orphan[4].

There are four basic levels of restriction for a GO part-of relationship defined [5]:

1. The part-of relationship makes no assumption of the existence of the relationship between the nodes in either direction. The child may or may not be part of the

[4]The GO curators are undertaking an effort to remove such orphans – personal communication from Amelia Ireland.

parent and the parent may or may not have the part child.

2. Wherever the child exists, it is as part of the parent.

3. this is the exact inverse of type two; wherever the parent exists, it has the child as a part, but the child is not necessarily part of the parent.

4. This interpretation is a combination of both 2 and 4[5], " 'has-part' and 'is-part'. An example of this is 'nuclear membrane' is part-of 'nucleus'. So 'nucleus' always has-part 'nuclear membrane', and 'nuclear membrane' is always part-of 'nucleus'."

The true path rule states that 'the pathway from a child term all the way up to its top-level parent(s) must always be true'. This should be "true" for both kinds of relationship in GO. An instance labelled as `Man`, could also legitimately be labelled as `Person`. So, a gene product labelled as having the function 'photoreceptor' is also a kind of 'signal transducer' and finally, has 'molecular function'. Thus, the true path rule, when working along `is-a` relationships folows a standard 'subclass' semantics, where all instances of `Man` are also instances of `Person`.

The GO editing style guide mentions that the majority of part-of links in GO are of type two. Types one and three are not used as they would violate the 'true path' rule in GO [5]. The part-of link is transitive and it is useful to note that the transitivity works along both kinds of relationship simultaneously, so that a path using both `is-a` and `part-of` becomes `part-of`.

One of the implications of the 'true path' rule is that the type of part-of relationship used in GO, outlined above, is restricted to those types where a child term must always be part-of its parent (the second rule). The GO editing style guide states that adding a new part-of relationships may break this rule. In this case, it states that the best strategy is to re-structure GO with new nodes and relationships so that the true path rule is followed.

Given the DAG representation, inverses of relationships do not occur and if they did, would/may violate the "true path" rule. So, while we might have stated that a `Testis` is part of `Man`, we know nothing about if a `Man` has part `Testis`, unless rule four is invoked. Unfortunately, the DAG contains no information as to which rule is to be invoked.

5 Reconciling the Two Representations

The DAG `is-a` relationship and OWL subsumption seem simple to reconcile; they have the same semantics: every instance of a class is also an instance of it's superclass.

We can assume that subclasses in the DAG representation, like OWL subclasses, overlap by default. It is, after all, not uncommon to see multiple inheritance in the GO DAG. This will capture most of the biology in GO correctly. However, we need to ask the question (and similar variants for each GO aspect): Is it possible for an individual molecular function to be both function-x and function-y at the same time? This is not the same question as a protein having multiple functions. The GO DAG does not

[5]The GO usage guide says three and four.

capture disjointness and its inclusion is a matter of capturing biological knowledge, so there is no way of simply automating knowledge of disjointness.

5.1 Capturing `part-of` in OWL

OWL provides an expressive language that allows us to explicitly state how relationships should be interpreted. Section 4 discusses four possible interpretations of GO's part-of relationship, and we show here how these different interpretations can be captured via alternative translations to OWL axioms. The advantage here is that rather than using a single construct which may be interpreted in a number of different (and in some sense *implicitly* specified) ways, the representation tells us precisely how to interpret the particular relationship. We can then have different instances of the part-of relationship (*e.g.*, those discussed in Section 4) interpreted using different levels of restriction, without any danger of confusion. In the following examples, we consider how we capture the particular semantics of the assertion `A part-of B`.

Type 1 These restrictions cannot be accurately captured in OWL as they deal with the *potential* for the relationship. OWL is designed to capture *facts*.

Type 2 Whenever the child exists, it is part of the parent. This can be represented through the following axiom:

SubClassOf(`SemiNiferousTubule` restriction(`part-of`
someValuesFrom(`Testis`)))

stating that for any instance of `Semi Niferous Tubule`, there *must* be an instance of `Testis` of which it is a part.

Type 3 Whenever the parent exists, it has the child as part. This can be represented through the following axiom:

SubClassOf(`Testis` restriction(`has-part`
someValuesFrom(`Semi Niferous Tubule`)))

Note that in this case we require the use of a property `has-part` which is defined as the inverse of `part-of`[6]. Inverse properties are interpreted as one would expect – if ever two individuals `Semi Niferous Tubule` and `Testis` are related via a property, then `Testis` and `Semi Niferous Tubule` are related via the property's inverse. The axiom above then states that for any instance of `Testis`, there *must* be an instance of `Semi Niferous tubule` that it has as a part.

Type 4 This is simply a combination of 2 and 3, and we can encode this by including both axioms introduced above.

[6]Many Description Logics allow the definition and use of inverse relationships. In OWL, there is no inverse property *operator* for use in expressions, but we can introduce and define properties as inverses.

SubClassOf(`SemiNiferousTubule` restriction(`part-of`
someValuesFrom(`Testis`)))
SubClassOf(`Testis` restriction(`has-part`
someValuesFrom(`Semi Niferous Tubule`)))

In OWL-DL, all classes are kinds of other classes – everything is a kind of something. Thus, orphans do not exist and unless otherwise stated, a class would be a subclass of a special class, named `Top`. In a GO DAG to OWL-DL conversion, we make orphan nodes subclasses of the the local root class (*e.g.*, `Molecular-Function`. So, `Plastid` is part-of the `Cytosol`, but has no `is-a` parent. The GOC is, however, undergoing a process of inserting appropriate `is-a` parents, so in the near future, this will no longer be an issue[7].

We can see, therefore, that it is possible to represent what is captured in the GO in OWL-DL. The OWL representation will capture the same knowledge as the Gene Ontology. In addition, we can even distinguish between the uses of rules two and four in the part-of relationship in GO. OWL cannot, however, represent the DAG's rule one, because OWL only represents facts, not possibilities. Finally, the only issue left unresolved is that of OWL's open world assumption and whether the DAG has a closed or open world assumption. Space does not permit an exploration of this aspect.

6 Conclusion

We have seen that converting from DAG to OWL presents no real problem, but is this true of the conversion back to OWL? One issue pertinent to the GO itself are the use of relationships other than `is-a` and `part-of`. GO only permits these two relationships, so conversion back again would be lossy. We also have the general loss of expressivity in the conversion back to DAG representation. In the conversion, we have to ensure that the true path rule is followed. This might well involve restrictions on the ontological constructs used.

The DAG style of ontology is static, with the ontology *pre-enumerated* and the only classes that can exist are those already present. A DL ontology, in contrast, is dynamic or *post-enumerated*[1], i.e., we can instantiate objects of classes that we make up on-the-fly. That is, we can use complex class descriptions to describe objects, and these descriptions are then taken into account by the reasoner. Thus, in DL style ontologies, it is common for classes defined in an ontology to be the building blocks of other classes, rather than enumerating all the possible classes[8]. Obviously, converting a dynamic ontology to a static ontology, without performing this enumeration might lead to an impoverished ontology.

In summary, we have described the role of the semantics of a representation language. The core of the argument is that if ontologies are to fulfil their role of providing a common, shared understanding of a knowledge domain, then the statements within that ontology have to be able to be interpreted unabmiguously. We then examined the semantics of GO's DAG as described above; the conversion the other way might

[7] Personal communication from Amelia Ireland.

[8] It is obviously desirable that the ontology only implies correct classses and would not include all possible classes by simple combination.

present more issues. The results of examining the semantics of the two reprsentations is, however, that conversions between the two authorised OBO representations is possible (within some constraints). Despite the apparent mundane nature of language semantics, it is at the core of ontology development and use and presents a useful tool in ontology re-use.

References

[1] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook Theory, Implementation and Applications*. Cambridge University Press, 2003.

[2] Michael Bada, Robert Stevens, Carole Goble, Yolanda Gil, Michael Ashburner, Judith A. Blake4, J. Michael Cherry, Midori Harris, and Suzanna Lewis. A Short Study on the Success of the Gene Ontology. Accepted for publication in the Journal of Web Semantics, 2004.

[3] Gene Ontology Consortium. Gene Ontology: Tool for the Unification of Biology. *Nature Genetics*, 25:25–29, 2000.

[4] Gene Ontology Consortium. Creating the Gene Ontology resource: Design and implementation. *Genome Research*, 11:1425–1433, 2001.

[5] Gene Ontology Consortium. The GO Editorial Style Guide, 2004. Available *via* `http://www.geneontology.org/docs/usage.html` May 2004.

[6] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[7] R. Stevens, C. Goble, I. Horrocks, and S. Bechhofer. Building a Bioinformatics Ontology Using OIL. *IEEE Transactions on Information Technology and Biomedicine*, 6(2):135–41, Jun 2002.

[8] R. Stevens, C. Goble, I. Horrocks, and S. Bechhofer. OILing the Way to Machine Understandable Bioinformatics Resources. *IEEE Transactions on Information Technology and Biomedicine*, 6:129–34, Jun 2002.

[9] C.J. Wroe, R.D. Stevens, C.A. Goble, and M. Ashburner. A Methodology to Migrate the Gene Ontology to a Description Logic Environment Using DAML+OIL. In *8th Pacific Symposium on biocomputing (PSB)*, pages 624–636, 2003.

[10] Iwei Yeh, Peter D. Karp, Natalya Fridman Noy, and Russ B. Altman. Knowledge acquisition, consistency checking and concurrency control for Gene Ontology (GO). *Bioinformatics*, 19(2):241–248, 2003.