School of Computing

Scaling the development of large ontologies: Identitas and Hypernormalization

Nizal K. Alshammry

Submitted for the degree of Doctor of Philosophy in the School of Computing Science, Newcastle University

April 2021

ABSTRACT

During the last decade ontologies have become a fundamental part of the life sciences to build organised computational knowledge. Currently, there are more than 800 biomedical ontologies hosted by the NCBO BioPortal repository. However, the proliferation of ontologies in the biomedical and biological domains has highlighted a number of problems. As ontologies become large, their development and maintenance becomes more challenging and time-consuming. Therefore, the scalability of ontology development has become problematic. In this thesis, we examine two new approaches that can help address this challenge.

First, we consider a new approach to identifiers that could significantly facilitate the scalability of ontologies and overcome some related issues with monotonic, numeric identifiers while remaining semantics-free. Our solutions are described, along with the Identitas library, which allows concurrent development, pronounceability and error checking. The library integrated into two ontology development environments, Protégé and Tawny-OWL. This thesis also discusses the ways in which current ontological practices could be migrated towards the use of this scheme.

Second, we investigate the usage of the hypernormalisation, patternisation and programatic approaches by asking how we could use this approach to rebuild the Gene Ontology (GO). The aim of the hypernormalisation and patternisation techniques is to allow the ontology developer to manage its maintainability and evolution. To apply this approach we had to analyse the ontology structure, starting with the Molecular Function Ontology (MFO). The MFO is formed from several large and tangled hierarchies of classes, each of which describe a broad molecular activity. The exploitation of the hypernormalisation approach resulted in the creation of a hypernormalised form of the Transporter Activity (TA) and Catalytic Activity (CA) hierarchies, together they constitute 78% of all classes in MFO. The hypernormalised structure of the TA and CA are generated based on developed higher-level patterns and novel *content-specific patterns*, and exploit ontology logical reasoners. The gen-

erated ontologies are robust, easy to maintain and can be developed and extended freely. Although, there are a variety of ontologies development tools, Tawny-OWL is a programmatic interactive tool for ontology creation and management and provides a set of patterns that explicitly support the creation of a hypernormalised ontology.

Finally, the investigation of the hypernormalisation highlighted inconsistent classifications and identification of significant semantic mismatch between GO and the Chemical Entities of Biological Interest (ChEBI). Although both ontologies describe the same real entities, GO often refers to the form most common in biology, while ChEBI is more specific and precise. The use of hypernormalisation forces us to deal with this mismatch, we used the equivalence axioms created by the GO-Plus ontology.

To sum up, to address the scalability and ease development of ontologies we propose a new identifier scheme and investigate the use of the hypernormalisation methodology. Together, the Identitas and the hypernormalisation technique should enable the construction of large-scale ontologies in the future.

Declaration

I declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research. No part of this thesis has previously been submitted for a degree or any other qualification at Newcastle University or any other institution.

Nizal K. Alshammry

April 2021

Publications

Portions of the work within this thesis have been documented in the following publications:

Alshammry, Nizal, and Phillip Lord. Identitas: A Better Way To Be Meaningless. *The 8th International Conference on Biomedical Ontology (ICBO 2017)*,

Alshammry N, Lord P. Identitas: Semantics-free and human-readable identifiers. Applied Ontology. 2021(Preprint):1-6.

Acknowledgements

First and foremost, I would like to thank God, the Almighty, for giving me the strength, patience and knowledge that allow me to complete this research.

Without the kind support and help of various people, this scientific research would be impossible to conduct.

I am genuinely thankful to my research supervisor Dr Phillip Lord, for his persistent guidance and invaluable supervision during my PhD study. At many stages during the thesis project, his enormous knowledge and ample experience have inspired me to solve many problems. My gratitude extends to my secondary supervisor Dr Paolo Zuliani who was keen to help me with my project.

In addition to my supervisors, I would like to thank Prof. Robert Stevens for his helpful advice and important insights at the start of the project.

Also, I am incredibly grateful to my parents, my wife and my children. Without their considerable support and sacrifices in the past few years, it would be impossible for me to complete the study. I am also grateful for all the help and support that I have received from my colleagues and friends.

Finally, I am highly indebted to the Northern Borders University, Saudi Arabia, for the funding and supporting my PhD study. Thanks also go to the staff of the School of Computing at Newcastle University for providing all the necessary support needed to carry out the research.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

ALC Attributive Language with Complements

API Application Programming Interface

ATP Adenosine Triphosphate

BPO Biological Process Ontology

CA Catalytic Activity

CCO Cellular Component Ontology

CWA Closed-World Assumption

ChEBI Chemical Entities of Biological Interest

CL Cell Ontology

DNA Deoxyribonucleic Acid

EC Enzyme Commission

EMBL-EBI European Molecular Biology Laboratory's European Bioinformatics Institute

F-logic Frame Logic

GO Gene Ontology

GOA Gene Ontology Annotation

GOOSE GO Online SQL Environment

GONG Gene Ontology Next Generation

GOC Gene Ontology Consortium

ICD International Classification of Diseases

IRI Internationalized Resource Identifier

IUB International Union of Biochemistry

JVM Java Virtual Machine

KR knowledge representation

KEGG Kyoto Encyclopedia of Genes and Genomes

MFO Molecular Function Ontology

OBO Open Biomedical Ontologies

ODPs Ontology Design Patterns

OIL Ontology Interface Language

OWL Web Ontology Language

OKBC Open Knowledge Base Connectivity

OORT OBO Ontology Release Tool

PATO The Phenotype And Trait Ontology

PRO PRotein Ontology

RDF Resource Description Framework

RO Relation Ontology

RIF Rule Interchange Format

SO Sequence Ontology

SNOMED CT Systematised Nomenclature of Medicine Clinical Terms

SHOE Simple HTML Ontology Extensions

SW Semantic Web

TA Transporter Activity

TCDB Transporter Classification Database

Uberon Uber Anatomy Ontology

W3C World Wide Web Consortium

XML eXtensible Markup Language

1

Introduction

Contents

1.1 Introduction

In recent years, biological and medical research has relied heavily on building standardised computational knowledge for many reasons. The goal is to be able to store, share, query, retrieve and analyse biological/biomedical data that often comes from complex experiments. Ontology as a computational modelling technique became a standard part of many scientific domains, including biological and biomedical domains, as it provides a common understanding of the fields through a semantic network of related concepts. As a result, many bio-ontologies have been developed and have become trusted source of information, such as Systematised Nomenclature of Medicine Clinical Terms (SNOMED CT) [106], International Classification of Diseases (ICD) [71] and the Gene Ontology (GO) [43]. In the meantime, ontologies development and management processes have become more complicated and challenging, especially with terminologies and ontologies of large-scale and complex domain of knowledge.

As ontologies become large, for instance, to reference concepts in the ontology becomes more challenging; the identifier scheme must scale, while maintaining ease of use. For example, GO and associated ontologies have adopted numeric identifiers, embedded in URLs that are made persistent through the use of PURL, a double-resolution mechanism. However, some ontological standards have raised concerns such as the use of numeric identifiers to uniquely identify ontology classes. Being able to address these issues by simplifying, clarifying or facilitating the processes of ontology development has been the goal of several methodologies and tools.

Scaling produces other challenges: after many years of development, hierarchies can become highly tangled and the possibility of errors (e.g., duplicate terms, incorrect classification) can increase. For instance, between July 2017 and July 2020, more than 14,700 new concepts were added to the international edition of SNOMED CT [1], and around 130,000 concepts have changed. This is mainly because of the fast growth in the biomedical domain. Any required development and maintenance to the SNOMED CT hierarchy required maintaining multiple classifications. As a result,

[1] https://www.snomed.org

a considerable number of methodologies and tools have been introduced to support the development and maintenance of the SNOMED CT structure, costing more than 9 million USD each year [31]. One effort made by the SNOMED community is reformulating the content of SNOMED CT into a more expressive language that enables consistency checks, that is Web Ontology Language (OWL) [88]. However, the size and complexity of SNOMED made the resultant OWL version of SNOMED inaccessible in ontology editors due to its size that causes memory problems [38].

Similarly, the scale and size of GO has increased, and its hierarchical structure has become more tangled, which has negatively affected the maintenance and development processes of GO. The GO project provides a comprehensive computational representation of biological systems in three respects: the molecular functions performed by genes and gene products, the cellular locations where the functions occur and the biological program carried out by multiple molecular functions. It is a widely used model and a valuable source of references, as judged by the number of people and organisation that use it and published papers that cite it. Currently, GO has more than 45,000 classes [118] describing the different biological functions of gene products over three large sub-ontologies. The changes in GO occur on a daily basis. This mostly involves creating new classes and new relationships and removing incorrect classes and inconsistent relationships. Statistically, between Oct 2018 and Dec 2020, 790 new classes were created, more than 911 classes were merged, and more than 791 classes were deleted (see Figure 1.1). The Gene Ontology Next Generation (GONG) [120] project was an early attempt that translated the hand-crafted taxonomies of GO into formal and logic-based taxonomies using the DAML+OIL [60] ontology language and DL-based reasoners. Another effort to enhance the representation of GO was made by Mungall [84], who converted the textual definitions of the GO classes into computable logical definitions using classes from related external ontologies, mostly from the Open Biomedical Ontologies (OBO) Foundry ontologies (e.g., PRotein Ontology (PRO) [100] and Chemical Entities of Biological Interest (ChEBI)). Within the Gene Ontology Consortium (GOC) the development of tools that facilitate the construction and maintenance of GO have increased to achieve several objectives, such as increasing expressivity, facilitating development

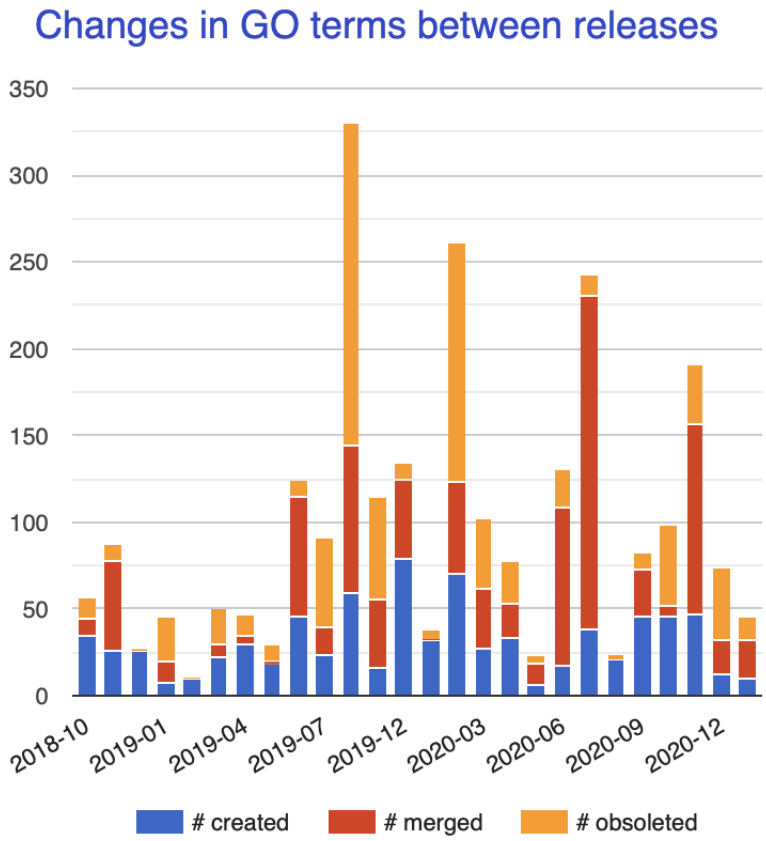


Figure 1.1: The changes in GO terms during the last two years. Figure taken from http://geneontology.org/stats.html, used under CC-BY 4.0 license.

and maintenance processes, and strengthening robustness.

In this thesis, we investigate the use of the hypernormalisation methodology [77] by reconstructing the Molecular Function Ontology (MFO), a sub-ontology of GO. Hypernormalisation is an extension of normalisation methodology [93]. Ontology normalisation aims to facilitate the development, reusability and maintenance of ontologies by creating explicit and modular ontologies. The methodology is divided into two independent stages: "ontological normalisation" refers to the basic knowledge structure of ontology concepts that are to be defined cleanly using some technique such as OntoClean. The second part is the normalisation of ontology implementation that can be achieved by disentangling the structure of an ontology into disjoint taxonomies: *self-standing classes* and *refining classes*. The *self-standing*

classes refer to the core concepts of a domain, that is, the physical and conceptual things in the world (e.g., "people","phone") that may or may not be defined using other descriptive concepts and conditions. The second taxonomy is about the refining classes (e.g., "small, medium, large") that are used to define the classes in the first taxonomy, the *self-standing classes*. The relationships between the two taxonomies are determined using definitions, axioms and reasoners. The Galen Ontology is one of the biomedical ontologies that was built using the normalisation principles [92].

The hypernormalisation technique can be distinguished from the normalisation technique in the way that it makes the asserted hierarchy nearly or completely a flatten hierarchy among *self-standing classes*. Moreover, it relies heavily on the use of ontology designed patterns (ODPs) and ontological reasoners to facilitate the development and creation of the ontological hierarchies. In order to be able to implement this type of ontology development, it is helpful to have an ontology development environment that supports and provides higher-level patterns; for example, Tawny-OWL [76]; a fully programmatic interactive environment for ontology creation and management and provides a set of patterns that explicitly support the creation of a hypernormalised ontology. The amino acid ontology [107] was built using the hypernormalisation technique. By contrast, representing the Karyotype Ontology in hypernormalised form was not possible [112]. So far, the ontologies that were built using the hypernormalisation principles are relatively small ontologies with 500 or fewer ontological classes.

In this thesis, we investigate two issues that address the scalability and ease of development of large ontologies.

First, we describe a new identifier scheme for generating local identifiers. This raises our first research question:

RQ1 What is the advantages of this approach to ontology development?

Second, we investigate the usage of hypernormalisation, patternisation and programatic approaches by asking how we could use this approach to rebuild the Gene Ontology, specifically the MFO.

The research questions that we are trying to answer, using the hypernormalisation, pattern-driven development approach is:

RQ2 Is it possible to apply this approach and what we do learn from exploiting the hypernormalisation and patternisation methodologies for a large ontology, using the MFO as a case study?

After applying the hypernormalisation and patternisation methodologies, three questions will be raised:

RQ2.1 How is the shape of the ontology changed from a denormalised ontology?

RQ2.2 Does the process of hypernormalisation teach us anything about the semantics and representation of the existing ontology.

RQ2.3 What kind of query capability do we get from a large hypernormalised ontology which is not possible with non-normalised ontology?

Finally, considering both hypernormalisation and identification:

RQ3 In using this methodology for an ontology of this size, do we ease the development and maintenance?

1.2 Contributions of this thesis

This thesis aims to address the challenges of scalability and to ease of development of large ontologies. Firstly, we implemented a new scheme for generating local identifiers, which enables the development of identifiers that are semantics-free, can be read by humans and checksummable. Secondly, we explored the usage of hypernormalisation and pattern-driven development approach by rebuilding the MFO. The investigation of the hypernormalisation approach resulted in the development of a hypernormalised form of the Transporter Activity (TA) and the Catalytic Activity (CA) hierarchies. Moreover, the investigation led to identification of significant semantic mismatch between GO and ChEBI.

The first contribution, which addresses RQ1, is represented in the Identitas library, a new approach to identifiers that could significantly improve the management of ontologies and overcome some related issues with monotonic, numeric identifiers, while remaining semantics-free. In our project, we have implemented a new scheme, which enables the development of identifiers that are semantics-free and can be read by humans. In this contribution, we have demonstrated that the scheme is applicable and scales easily to befit the size of current ontologies. We have also considered ways in which current ontological practices could be migrated towards the use of this scheme. Finally, Identitas has been integrated into environments for ontology development such as Tawny-OWL and Protégé.

To answer the RQ2, we rebuild the MFO using the strategy we describe in Chapter 5, starting with transporter activity and then catalytic activity.

During the reformation of the transporter activity structure, we identified three main categories that comprise the hierarchy of TA, that is, general transporter classification classes, active transporter classes and facilitated diffusion classes. We developed these classes using *content-specific patterns*(CPs) [41], that were built with a set of biological properties to describe each specific transporter category. Like in software engineering, Ontology Design Patterns (ODPs) have become a key component of ontologies development lifecycle as they allow the creation of a rich and replicable representation of an ontology, reducing the effort and time spent during an ontology design process and easing communication between ontology developers [42]. In this contribution, we demonstrate that it is possible to represent the GO transporter activity in a hypernormalised form by disentangling its structure into the two independent disjoint taxonomies *self-standing classes* and *refining classes* that we introduced earlier. Using the high-level patterns provided by [77] allow the explicit and accurate construction of TA hierarchies.

Similarly, we have also applied the methodology on the catalytic activity, the broadest grouping class in MFO because it includes the largest number of classes. The investigation shows that 75 percent of the CA classes describe chemical reactions catalysed by different enzymes using a chemical equation. Therefore, we discuss different modelling solutions that enable the creation of chemical equations using

computable logical definition, which need to address several challenges such as: directionality of reactions; reactions stoichiometries; and support the automatic inference of relationships. The designed logical definitions support the automatic inference of memberships by relying greatly on the ChEBI classification of chemical entities with the assistance of the Rhea database as an intermediary.

Regarding the RQ2.1, RQ2.2 and RQ2.3, we show that it is possible to represent the GO transporter activity and catalytic activity in a hypernormalised form, the developed classification is robust, easy to maintain and can be developed and extended smoothly. As with the Gene Ontology, the hypernormalised hierarchy will have incorrect classification, however, we find them to be less error-prone for a number of reasons. In the hypernormalisation of TA classes and CA equation classes we did not assert subsumption relationships among the *self-standing classes*, instead we built domain-specific logical patterns and exploited logical reasoners to build the relationships among the classes using **Hyper-GO** refining classes and equivalent classes from external ontologies and their ontological classification. As a result, the hypernormalised hierarchies inferred a lot of the same classifications that exist in the denormalised hierarchies, but also showed a number of differences. The two reasons for these differences are: the mismatch in semantics between GO and ChEBI; a new relationships that have been inferred that were not captured in the denormalised hierarchies. Lastly, by converting the textual definitions of the GO classes into computable logical definitions we have increased the ability to query expressively.

We believe that the wide use of random identifiers would enable concurrent development of ontologies, especially when using large enough identifier space. With Identitas we have combined this readability and checksummability. Similarly, we have shown that the exploitation of our hypernormalisation methodology resulted in the construction of explicit, manageable and robust TA and CA ontologies using higher-level patterns and logical reasoners. Taken together we believe this addresses RQ3, which should enable the large-scale construction of ontologies in the future.

1.3 Thesis structure

This thesis consists of the following chapters:

- In Chapter 2 we answer essential questions related to ontologies: the motivation for using ontologies; the definition of an ontology; the set of formal languages in which ontologies can be expressed; the most successful ontologies; the different technologies that have been used to create ontologies; and the applications that have been made, which is mostly taking place in the domain of bio-medicine. This chapter ends with specifying the terminologies that we will use throughout this thesis.

- The Tawny-OWL library is presented in Chapter 3. It is the main tool that was used to develop our **Hyper-GO** ontology. We describe the core functions of Tawny-OWL, its advantages over traditional tools and emphasise the importance of higher-level patterns for building a hypernormalised ontology. As Tawny-OWL is built on Clojure, we briefly introduce Clojure. We use examples from the rewritten Family Ontology[2] to show the way that OWL entities and axioms are defined in Tawny-OWL.

- In Chapter 4 we introduce the Identitas library, a new style of identifiers dedicated to the notion of semantic-free IDs and implements the following features: concurrent development, pronounceability and check for errors. We also show that the scheme is applicable and can scale easily to the size of current ontologies, including the Gene Ontology, as well as considering ways in which current ontology practices could be migrated toward the use of this scheme. A version of this chapter has recently been accepted for publication in the Applied Ontology Journal [7].

- In Chapter 5, we investigate the ontological representation of the molecular functions defined in the MFO. In that chapter, we discuss the steps to hypernormalise the molecular activities, and we identify inconsistent classifications.

- In Chapter 6, we show how transporter activity classes are currently structured in GO, and how they can be hypernormalised using content-specific and higher-level patterns, which support the creation of a hypernormalised ontologies.

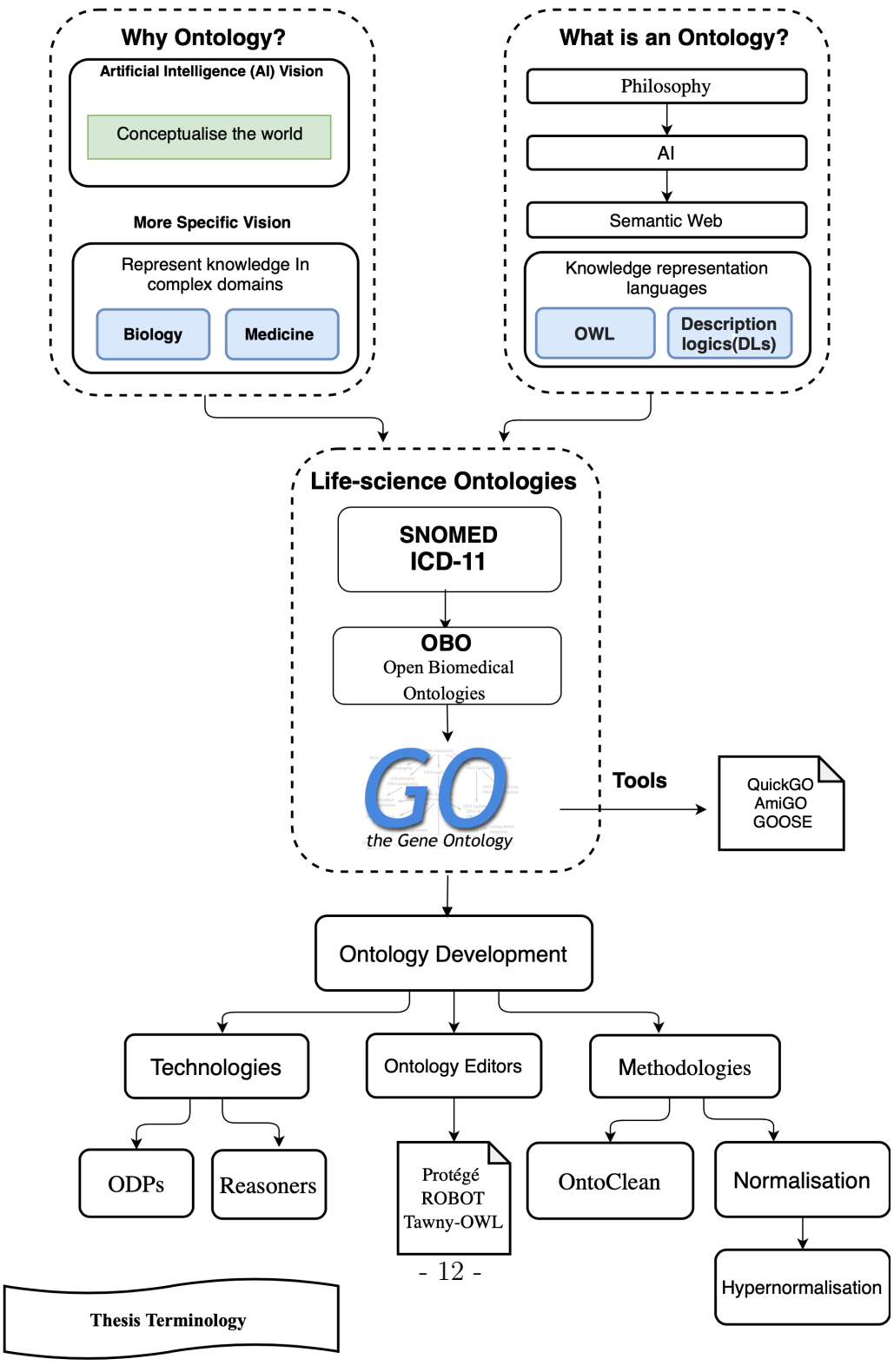
[2]Family Ontology is an OWL Prime ontology rewritten in Tawny-OWL, available at `https://github.com/phillord/owl-primer`

- In Chapter 7 we show how catalytic activity classes are currently structured in GO, and we discuss how to redefine these classes using logical definitions. In addition, we discuss alternative solutions that are often simpler but are less meaningful and probably do not support the correct and automatic inference of relationships.

- In Chapter 8 we discuss the outcomes of this thesis and how using Identitas, hypernormalisation and patternisation techniques helps to address the scalability and ease of development of large ontologies.

2

Background and Related Work



2.1 Ontology

This section is largely intend to answer essential questions related to ontologies: the motivation for using ontologies; the definition of an ontology; the different technologies that have been used to create ontologies; and, the applications that have been made which is mostly taking place in the domain of bio-medicine.

2.1.1 Why Ontology?

Ontologies have gained popularity in the fields of computer science and information science because of the need for a mechanism to identify and represent complex structured knowledge, which is a necessary step towards making intelligent systems [24]. Although, there are several kinds of modelling techniques, such as relational databases, ontologies provide a rich classification and description of reality. Some of the key characteristics of ontologies include the provision of a common terminology, which facilitates better understanding of a domain's content, knowledge re-usability, and shareability. Together, these factors increase the level of interoperability among different systems or databases [8]. Originally, the vision of the Artificial Intelligence (AI) community was to produce a generalised description of the world; however, this was not achieved as the world is extremely complicated. Knowledge of the world comes from various sources (often from different fields), which may not have the same classification of the universe.

A narrower ambition was representing knowledge in complex domains with rich content, like biology and medicine, where ontologies have now become widely used and a very common model for how to structure data. Accordingly, many ontologies have since been developed and become trusted sources of information across several scientific domains. For instance, the Systematised Nomenclature of Medicine Clinical Terms (SNOMED CT) [106] and International Classification of Diseases (ICD) are two widely accepted classification systems which provide a comprehensive and standardised health terminologies that cover most areas of medicine.

2.1.2 What is an Ontology?

The term "ontology" was originally introduced as a related subfield of the metaphysics branch (known as general metaphysics) of philosophy, which concerns the

study of the entities that exist in the universe, or are posited to exist, and their categories [19].

The difference between ontologies and metaphysics is that the former is intended to answer questions about things that exist in the world, while the latter attempts to answer questions about how do things exist. In the 1970s, researchers in the AI community recognised that the way to build and maintain strong intelligent information systems is using ontologies as computational knowledge models. In the early 1990s, ontologies gained momentum in computer science because of the widely discussed paper of Tom Gruber, which defined an "ontology" as "an explicit specification of a conceptualization [1]" [46]. As the use of ontologies became widespread in the computer science community, the definition transformed to include more adjectives describing specification and conceptualization (such as "formal", "shared"). A well-known modern description for an ontology is *"an explicit and formal specification of a shared conceptualization in the area of interest"* [79]. That is to say, for ontologies to provide a representational machinery model for domain knowledge, they need to be specified formally, just as a computer program is formally encoded by programming language. Moreover, to enable knowledge sharing and reuse, there should be a level of agreement on the usage of vocabulary in the domain of discourse.

In this thesis, first, we clarify the difference between an ontology and other types of conceptual specification schemes, and give a brief history of formal ontology languages. Ontologies are more complex than other existing techniques that use formal specifications to represent knowledge such as taxonomies, thesaurus, and controlled vocabularies. A controlled vocabulary is a restricted list of terms that do not necessarily have relationships to each other or possess a specific structure. A taxonomy is a specific kind of controlled vocabulary, and the simplest technique with which to organise terms into a hierarchical structure using limited types of relationships. A taxonomy is usually displayed in the form of a tree structure, while a thesaurus includes additional relationships between terms (e.g., "see also" relationship) in a standard structure. Despite the fact that a taxonomy is the backbone of an on-

[1] Gruber identified conceptualization as "an abstract, simplified view of the world that we want to represent for some purpose."

tology, an ontology can have more meaningful relationships among domain entities, and relationships specific to the content, not only a parent-child relationship. In other words, the relationships among entities are multidimensional relationships, which are reflected on the overall representation of the domain of interest. As such, they are suited to representing more complex domains like medicine, and add new knowledge using restrictions, rules, and axioms. Yet, though ontologies are not limited to pre-specified types of relationships, this does not necessarily imply that everything can be represented by an ontology. For a domain knowledge to have a concrete representation and processed automatically in computers, it has to be represented using an ontology or formal knowledge representation (KR) languages. There are a number of formal languages designed to express ontologies for a particular domain, such as Ontology Interface Language (OIL), Simple HTML Ontology Extensions (SHOE), Knowledge Interchange Format (KIF), Frame Logic (F-logic) and Web Ontology Language (OWL). Ontology languages classified to several different categories: frame-based languages (such as F-logic and Open Knowledge Base Connectivity (OKBC)), description logics-based languages (DLs) (such as KL-ONE and OWL), First-order logic-based languages (FOL) (such as KIF and CycL) and web-based languages (such as eXtensible Markup Language (XML) Resource Description Framework (RDF), RDF Schema, OWL) [30]. However, there are some languages which belong to more than one categories. For instance, the OWL [88] is a web standard language which extends the semantic interpretation of the RDF syntax and based on a low level knowledge representation technique that is description logic (DL).

Although, there are several languages in which ontologies can be expressed, most modern ontologies share common elements to define the knowledge in a domain of discourse [75]:

1. Classes (also called: 'concepts'): primary components of an ontology structure; they represent sets of collections of entities within a domain, which share common attributes. For example, Deoxyribonucleic Acid (DNA) is a concept in the scientific domain of molecular biology.

2. Relation: the way to link related individuals of a domain to each others. For example, DNA `is composed of` smaller molecules called nucleotides.

3. Instances (also called 'individuals', 'particulars'): the base components of an ontology; they represent the entities which belong to a class in the domain of interest. For example, a particular piece of Nuclear DNA located in the nucleus of a eukaryotic cell. Although instances are the base components of an ontology, they are often not represented explicitly in an ontology.

4. Attributes (also called 'properties', 'slots'): refer to the characteristics or properties that an entity may have. For example, most DNA molecules have the shape of a double helix and some are 2m in length.

Next, we consider the advantages of OWL and DL based languages. The use of DLs originated from the need to add formal and logic-based semantics to the existing semantic networks and frame-based approaches. DLs comprise a successful family of KR formalisms, which provide a set of formal and logic-based language constructs that can be used to describe domain knowledge concepts in terms of their properties and relationships with other concepts. DLs' basic building blocks are atomic concepts (e.g., Human, Male, Female), atomic roles (e.g., marriedTo, hasChild), and individuals' names (e.g., JOHN). However, more complex concepts can be defined using language constructors. Therefore, the expressivity of the DLs languages depends on the enabling of different constructs in their languages, such as conjunction ($\sqcap$), disjunction ($\sqcup$), negation ($\neg$) and quantifiers ($\exists, \forall$). For example, the description logic Attributive Language with Complements (ALC) provides the basic language constructors which later being extended by other languages of the DL family, such as SHIQ that added more constructors to express transitive, inverse roles, and cardinality restrictions. Another key feature of DLs is their amenability to automated reasoning. A description logic reasoner (also known as classifier) is software tool which interprets the description logic rules to a set of axioms (true statements) to deduce indirect relationships between domain concepts. It is an important and valuable tool for the purpose of checking an ontology consistency, inferring any implicit relationship between specific domain concepts from explicit

defined relationships, and ensuring correctness of classification outcomes. However, as expressive languages become very popular the computational reasoning with such languages become a challenging task. One of the most expressive languages is OWL which uses description logics as its underlying logic foundation.

The Web Ontology Language (OWL) is a Semantic Web (SW) technology designed by the World Wide Web Consortium (W3C) as a formal language for encoding knowledge on the Web. The W3C vision of the SW is to make information on the Web computationally processable by computers instead of only displaying information interpretable by people. To facilitate this process, ontology languages have played a vital role in linking data on the Web by precisely defining the structures of knowledge for various domains using formal syntax and semantics. The OWL language has become a standardised and broadly accepted ontology language as it provides far richer and logic-based semantics than those of the earlier Semantic Web languages initiatives XML, RDF, and RDFS. These languages or technologies form the SW architecture and are built on top of each other, as shown in the SW layer cake diagram (also known as the SW stack) of Figure 2.1. At the bottom of the SW stack are the URI/IRI technologies, adopted from the hypertext Web to uniquely identify resources on the SW. The XML language provides a standard syntax to enable the creation of documents of structured data. At present, XML is not the only syntax as there are alternative syntax with different features. RDF is the first language developed to semantically describe resources on the Web, which it does by relying on Uniform Resource Identifiers (URIs) and using a triple-based format (i.e., subject-predicate-object). SPARQL is a data query language developed to query over RDF and any RDF-based data. For relationships that cannot be directly described using description logic, logical rules can assist with this, as long as they are defined using Rule Interchange Format (RIF). RDFS extends the basic RDF vocabularies to allow resources to be classified into classes and subclasses, and imposes restrictions on properties using vocabulary such as `rdfs:class`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, and `rdfs:range`, which allow the creation of simple ontologies. However, RDFS is still not greatly expressive; it lacks a sufficient number of sets of vocabularies to represent the essential relationships between the

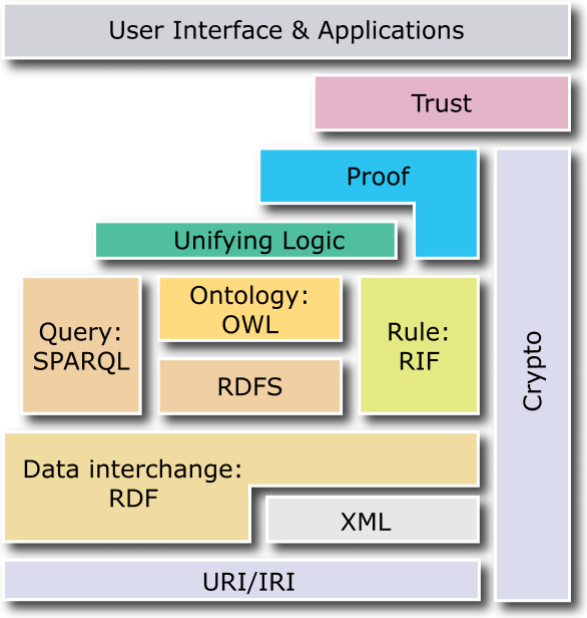


Figure 2.1: Latest Semantic Web Layer cake diagram, from `https://www.w3.org/2001/sw/`

classes of complex domain knowledge. In this regard, OWL offers the ontology builder an additional rich vocabulary for expressing important domain classes and properties, and adds constraints on the use of this vocabulary. On the top of the ontology layer, there is the unifying logic layer, which is responsible for assuring the consistency and correctness of the interchanged data. Once their unified logic is assured, the proof and trust layers then evaluate the quality of the data and ensure their trustworthiness, before transferring the final results to a user interface and applications.

To introduce some of the key language features that OWL offers, we provide examples from OWL Primer[2], which can be encoded using several syntactic formats, though here, we use the standard OWL2 syntax RDF/XML [40]. In addition to RDFS vocabularies, OWL has introduced other advanced language constructors to specify the relationships between classes, such as `DisjointClasses` and `EquivalentClasses`. For instance, though they can identify as neither, an individual cannot identify as both a man and woman. This can be encoded using `DisjointClasses` (see Listing 2.1) to allow OWL reasoners to check if developers have mistakenly classified an individual to be a member of (or more technically "an

[2] `https://www.w3.org/TR/owl2-primer/`

instance of") both classes, `Man` and `Woman`.

```
<owl:AllDisjointClasses>
  <owl:members rdf:parseType="Collection">
    <owl:Class rdf:about="Woman"/>
    <owl:Class rdf:about="Man"/>
  </owl:members>
</owl:AllDisjointClasses>
```

Listing 2.1: An example of using the OWL DisjointClasses feature.

Conversely, there are things in world that refer to the same groups and they are semantically equivalent, as in the case of person and human. Every individual that is classified as a person, is also a human, and vice versa (see Listing 2.2).

```
<owl:Class rdf:about="Person">
  <owl:equivalentClass rdf:resource="Human"/>
</owl:Class>
```

Listing 2.2: An example of using the OWL EquivalentClasses feature.

In addition, OWL allows you to apply restrictions on the use properties in order to describe a group of individuals which satisfy those restrictions. The following example states that a woman may have at most one husband using the cardinality constraint `maxCardinality` (see Listing 2.3).

```
<owl:Class rdf:about="Woman">
 <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="#hasHusband" />
    <owl:maxCardinality rdf:datatype=
          "&xsd;nonNegativeInteger">1</owl:maxCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Listing 2.3: An example of using the OWL property cardinality restrictions feature.

Other advanced modelling capabilities of OWL that increase the language expressivity are the logical characteristics of properties, namely, transitive, functional, inverse functional, reflexive, irreflexive, disjoint, symmetric, asymmetric and inverse properties. These add more semantics to the use of properties and enable reasoners conformant with OWL to infer further information about resources. In case a property stated to be a transitive, and the property such as `hasAncestor` linked individual A to another individual B, which has the same relation to individual C, then the transitive property would enable a reasoner to infer that `A hasAncestor C`.

While, a symmetric property can only specify a relationship in one direction, and rely on a reasoner to infer the opposite direction of the relationship, for instance, `isMarriedTo` (in Listing 2.5). Conversely, a reasoner cannot infer the inverse relationship of the property of asymmetric type, as in the case of `hasChild` property which cannot be applied on both direction of individuals. Moreover, the same notion of classes disjointness can also be applied on properties in order to represent some common law in relationships. For example, an individual cannot have both relationships `hasParent` and `hasSpouse` to another individual. In Listings 2.4 to 2.6 we show some of the OWL property characteristics syntax. For a more detailed on property characteristics please refer to the OWL Prime web page[3].

```
<owl:TransitiveProperty rdf:about="hasAncestor"/>
```

Listing 2.4: An example of transitive property.

```
<owl:SymmetricProperty rdf:about="isMarriedTo"/>
```

Listing 2.5: An example of symmetric property.

```
<owl:AsymmetricProperty rdf:about="hasChild"/>
```

Listing 2.6: An example of asymmetric property.

```
<rdf:Description rdf:about="hasParent">
  <owl:propertyDisjointWith rdf:resource="hasSpouse"/>
</rdf:Description>
```

Listing 2.7: An example of disjoint property.

In general, the degree of expressivity and computational complexity of the OWL language varies among the three sublanguages of OWL: OWL Lite, OWL DL (Description Logic), OWL Full which are intended to fulfil the different needs of ontology developers.

1. OWL Lite: the simplest semantics version (compared to other versions of OWL) which is more suitable for domains with straightforward knowledge representation and limited cardinality restrictions. It is used to convert simple classification hierarchies, such as taxonomies into ontology format. OWL lite

[3] `https://www.w3.org/TR/owl2-primer/`

is a light version of the OWL DL and has semantic equivalent to description logic SHIF.

2. OWL DL: this specific language of OWL provides a high degree of expression in parallel with a guarantee to maintain computational completeness. The OWL DL syntax is mapped to the formal semantics of the description logic SHOIN(D). These semantics are fully decidable, but with a higher computational complexity than the semantics of OWL Lite. Table 2.1 shows the main language constructors integrated in OWL DL with the equivalent *SHOIN(D)* syntax.

3. OWL Full: although it uses the same set of language constructs as the OWL DL, OWL Full has few constraints in the way that these constructs can be used; as a result it does not map to a DL; this means that the automated reasoning over OWL Full is much less defined than for the other levels of OWL and would not necessarily be decidable.

The current version of OWL is the second edition (OWL2) [56] which was released in 2012 with the official RDF/XML syntax. In fact, the OWL language is specified in a high level structural specification that is then translated into several concrete syntaxes [121]. There are a number of syntaxes in which OWL ontologies can be expressed and shared, namely, RDF/XML, OWL/XML [57], the Functional-Style Syntax and Manchester Syntax [59]. Next, we provide a brief description of each syntax, their design purposes and any advantages or drawbacks associated with each syntax. Moreover, an example for each syntax is provided in Listings 2.8 to 2.11 with simple ontology name and class Thesis.

1. RDF/XML Syntax: defined by the W3C to be the standardised and default syntax for storing OWL2 ontologies by most OWL compliant tools. As the name implies, this syntax allow the encoding of RDF graphs in XML format. However, the syntax is extremely verbose and hard to read especially for complex knowledge representation (see Listing 2.8).

Table 2.1: Table summarises the basic building blocks and axioms that are used in constructing an ontology using OWL DL with the corresponding *SHOIN(D)*. The A and B denote either atomic or complex concepts (refer to as a class in OWL), c and d refer to individuals, and R and S are roles (refer to as a property in OWL) [104]

Constructor Name	OWL-DL	*SHOIN(D)* DL syntax
Conjunction	`intersectionOf(A,B)`	$A \sqcap B$
Disjunction	`unionOf(A,B)`	$A \sqcup B$
Negation	`complementOf(A)`	$\neg A$
OneOf	`oneOf(c,d,....)`	{c,d,.....}
Universal value restriction	`allValuesFrom(A)`	$\forall R.A$
Existential value restriction	`someValuesFrom(A)`	$\exists R.A$
Number (atleast) restriction	`minCardinality(n)`	$\geqslant nR$
Number (atmost) restriction	`maxCardinality(n)`	$\leqslant nR$
Number (exact) restriction	`cardinality(n)`	$= nR$
Concept inclusion	`rdfs:subClassOf(A,B)`	$A \sqsubseteq B$
Concept equivalence	`equivalentClass(A,B)`	$A \equiv B$
Property inclusion	`rdfs:subPropertyOf(R,S)`	$R \sqsubseteq S$
Property equivalence	`equivalentProperty(R,S)`	$R \equiv S$
Individual equivalence	`sameAs(c,d)`	$c = d$
Concept disjointness	`disjointWith(A,B)`	$A \sqsubseteq \neg B$
Individual disjointness	`differentFrom(c,d)`	$c \neq d$
Transitive property	`TransitiveProperty(R)`	$R^{+} \sqsubseteq R$
Symmetric property	`SymmetricProperty(R)`	$R \equiv R^{-}$
Functional property	`FunctionalProperty(R)`	$\leqslant 1R$
Inverse property	`inverseOf(R)`	R^{-}
Inverse Functional property	`InverseFunctionalProperty(R)`	$\leqslant R^{-}$

2. Functional-Style Syntax: this syntax is a text-based syntax and represents the link between a high level structural specification and various concrete syntaxes. Functional-Style syntax is used to define the semantics of the OWL2 ontologies and enable the mappings from and into exchange syntaxes, such RDF/XML syntax [88]. Although it is more human-readable syntax than RDF/XML, it is still verbose (see Listings 2.9).

3. OWL/XML Syntax: the primary advantage of using XML syntax to represent OWL ontologies is the use of XML processing and querying tools that are not compatible with RDF/XML, such as XPath and XSLT (see Listings 2.10).

4. Manchester Syntax: it is more human readable syntax, frame-based and being used in various ontology development tools, such as Protégé.

```
<rdf:RDF ... >
  <owl:Ontology rdf:about="http://www.example.com/ontology"/>
    <owl:Class rdf:about="...ontology.owl#Thesis">
</rdf:RDF>
```

Listing 2.8: RDF/XML Syntax.

```
Ontology(<http://www.example.com/ontology>
Declaration(Class(<http://www.example.com/ontology.owl#Thesis>))
)
```

Listing 2.9: Functional-Style Syntax.

```
<Ontology ....>
  <Declaration>
    <Class IRI="http://www.example.com/ontology.owl#Thesis"/>
  </Declaration>
</Ontology>
```

Listing 2.10: OWL/XML Syntax.

```
Ontology: <http://www.example.com/ontology>
Class: <http://www.example.com/ontology.owl#Thesis>
```

Listing 2.11: Manchester Syntax.

2.2 SNOMED-CT and ICD-11 Ontologies

Here, we introduce some of the most successful ontologies, which have developed based on description logics and using OWL languages. The features offered by Description Logics and OWL (i.e., well-defined semantics and automated reasoning) facilitate the construction of large and complex ontologies, and enable consistent representation of knowledge of many real-life application domains, such as biomedical and biological domains. For example, the early development of the medical model SNOMED-CT (abbreviated as SCT) depended mainly on the use of description logics and reasoning services to describe complex concepts related to clinical procedures, diseases, and treatments in classification hierarchies. This enabled the SCT community to implement more useful analytical operations, thus verifying the logical integrity of the model and subsequently improving the quality of clinical information. Currently, SNOMED-CT includes more than 355,000 classes, and has become the national standard in several healthcare institutions of various countries, such as the United Kingdom, United States of America, and Canada [103]. The SCT has concepts like *Procedure* (with a unique numeric identifier (`ID: 71388002`), which subsumes all the activities made to provide a health care, such as *Laboratory procedure* (with an `ID: 108252007`) and attributes, such as *Has focus* (`ID:363702006`) to specifies the focus of a specific procedure. The number of classes increases every year which make the processes of development and maintenance expensive and time-consuming. Moreover, the original formulation of SNOMED-CT has many limitations in terms of expressiveness such as the inability to state transitivity of properties or to determine the equivalence of concepts. The SNOMED-CT community considered reformulating the content of SCT into a more expressive language with consistent representation which was OWL. The transformation of SCT into an OWL ontology requires a Perl script provided by the SCT community to convert the RF2 files into OWL file [91]. The key advantages of this reformulation are: firstly, the current development of the SCT OWL version involves a large number of international medical experts who update the ontology using web-based collaborative platform, such as WebProtégé; secondly, improving the accuracy and expressivity of the classification; and thirdly, the large number of tools which support the cre-

ation, automated classification and visualisation of ontologies formalised using OWL syntax. It was not only SNOMED who made that decision, but also the 11th International Classification of Diseases (ICD-11) uses OWL to formalise the classification of patients diseases and health care conditions developed by the World Health Organisation (WHO). In fact, the SNOMED classifications have been used in the textual content of Electronic Healthcare Records (EHRs) to annotate these records which are then used to create the ICD-11 ontology. The reformulation enables ICD-11 to function in any health information systems, mostly EHRs and to link its concepts to other healthcare ontologies concepts such as SNOMED-CT. Representing SNOMED-CT and ICD-11 in OWL format supports semantic interoperability in health information systems because both ontologies use a common language in the description of same domain entities. However, ontologies have been used intensively over the last two decades, specifically in biomedical and biological research for different reasons which raises several issues such as the overlap of ontologies terms and reuse.

In the section that follows, we will discuss the substantial effort that has been made by the Open Biological and Biomedical Ontologies (OBO) Foundry [101] to manage issues associated with ontology development by providing a number of services and guidelines.

2.3 Open Biological and Biomedical Ontologies (OBO)

Open Biological and Biomedical Ontology (OBO, previously Open Biomedical Ontologies) consortium is a considerable effort that has been made to design a variety of reference ontologies which covers all areas in the biological and biomedical sciences. To achieve this goal, the OBO foundry provides a set of regularly updated principles, guidelines and best practices to be considered during ontology development by those who intend to submit their ontologies to the consortium. Several principles for ontology development, available at [4], include, ontologies are open, free and available to be used, ontologies authoring using common formal language (e.g.,

[4]http://www.obofoundry.org/principles/fp-000-summary.html

OWL, OBO format), orthogonal (i.e., each term is defined only in one ontology), ontologies have a unique space of identifiers (such as `GO`, `OBI`, `CL`) and have a content that is bounded to the stated scope. They use few capitalised words (such as MUST and SHOULD) to indicate when the principles are applied. For example, the principle Commitment To Collaboration means: ontology developers must provide a brief description about the submitted ontology and their assurances that there are no other ontologies cover the same domain. The claimed advantages of committing to the collaboration principle are: ensure reuse of ontologies terms (avoiding overlap), improving interoperability between different knowledge systems and built ontologies that are scientifically accurate and well-formed [101]. There are over 60 ontologies which have been retrofitted or built on the basis of OBO principles, such as the Ontology for Biomedical Investigations (OBI) [11]. OBI reuses terms from large number of OBO ontologies, specifically of biomedical knowledge representation, such as the Gene Ontology (GO) [43], The Phenotype And Trait Ontology (PATO) and Chemical Entities of Biological Interest (ChEBI) [52]. OBI includes more than 3,400 terms which describe the various elements involved in all phases of biomedical investigations, including design of experiments, procedures and devices used, biological material and the list of analysis performed on the acquired data.

In order to make better use of the principles, the OBO foundry built its own language along with the popular GUI-based ontology OBO-Edit software [32] for building biomedical ontologies. Originally [108] the OBO-format was less logically expressive than OWL, because it did not include all the constructs (e.g., restrictions on classes) which OWL has. However, the OBO format did include other standard syntax, which OWL only support as generic annotations, including synonyms (e.g., broad, exact, related, narrow), local identifiers and 'subset' construct (subset is a collection of terms only, and specified as part of an ontology). More recent versions of OBO format support all OWL constructs via extension mechanism using OWL functional syntax. Because the OBO guidelines gave developers the freedom to use whatever formats and technologies to build their ontologies, a number of OBO ontologies were developed using OWL. Conversely, several methodologies and tools have been reported to implement bidirectional OBO–OWL conversion, and

subsequently to enable semantic interoperability between Semantic Web and OBO systems. Recently, the OBO community support the use of ROBOT (OBO Tool) [62] that is a free and open source software, compatible with several ontology syntaxes (e.g., OBO format, OWL Manchester Syntax) and offers wide range of ontology development tasks, such as updating, testing, reasoning and format conversion.

The section below introduces the Gene Ontology (GO) [28], one of the most successful ontologies and its success that has inspired the development of a significant number of bio-ontologies and the creation of the OBO foundry. It is the focus of this thesis.

2.4 Gene Ontology (GO)

The Gene Ontology (GO) is the result of an intensive collaborative effort to provide a formal and computational representation of biological systems, including the functions of genes and genes products (e.g., proteins, RNA) from different organisms. One of the major goals of biomedical research is to understand the biological roles of the genes and their products at different molecular, cellular and organism levels. The benefit of studying the functional genomics of an organism is to gain enough experimental knowledge that can be applied to other organisms because a large set of the genes are shared among eukaryotic organisms [18]. There were several attempts aimed to interpret and annotate the function of genes, such as the use of natural language to describe data in biological databases which were found to be insufficient to accurately interpret the role of genes and to enable data integration [35]. Furthermore, there were several obstacles toward unifying the description of the function of genes, including the complexity of the domain knowledge, the significant growth of biological data that to be analysed and the diversity of molecular information about genes and their products in biological databases from different sources.

The Gene Ontology Consortium (GOC) addressees these challenges by providing formal, structured and consistent descriptions of the theory and practice of experimental biological knowledge concerning the function of genes [55]. The contribution of the GOC has been divided into three major efforts. The first contribution is the logical classification of the biological roles of genes and their relationships to other

functions, represented by the Gene Ontology (GO). GO is structured as a Directed Acyclic Graph (DAG), where each term (i.e., represent a gene function) placed in the DAG has a relationship, such as `is_a` or `part_of` to other terms in a related domain or different domains. Figure 2.2 shows part of the GO structure, where each GO term as node and the arcs represent the relationships among the terms across three ontologies. That is, the Gene Ontology composed of three large and non-overlapping sub-ontologies, namely Biological Process Ontology (BPO), Cellular Component Ontology (CCO) and Molecular Function Ontology (MFO). Each of the three independent ontologies covers key areas of the biological domain in which the functions of gene products take place.

- MFO: provides the set of terms that describe the different activities of gene products at the molecular level. Examples of a molecular-level activities are `catalytic activity, transport activity and binding activity`.
- BPO: contains a class of terms which describe a series of events which are achieved by one ore multiple ordered molecular activities. Example of biological process is `response to stimulus` which include more specific biological processes such as `cellular response to stimulus`.
- CCO: It contains the set of the terms that describe the parts of the cell, where the gene products reside. Example of cellular locations are: `nucleus, nucleolus and mitochondrion`.

The aim of the Gene Ontology is to represent the most current state of biological knowledge based on latest discoveries from published articles, methods and experiments [28]. Each month there is a new release of GO which includes significant improvements in the quantity and quality of the ontology. This includes defining new terms, adding additional relationships, rearranging ontology classification and linking gene products with the recent added terms that represent their functions.

The second contribution refers to the Gene Ontology Annotation (GOA) project that is an evidence-based collection of biological information created by defining a connection between a gene product and the related set of terms in GO based on

an evidence supporting that annotation [27]. These annotations are contributions from several model organism databases and biological research communities. In fact, the initial development of GO (in 1998) was a collaboration between three different models organism databases to interpret and annotate the biological functionality which genes and gene products contribute to the biological systems in only three organisms, namely, Mouse Genome Information(MGI) [16], Saccharomyces Genome Database(SGD) [26] and FlyBase [47]. Recently, the GOC has grown to include several other resources (available at [5]), such as the plant repository, animal repository and microbial genomes. Moreover, the GO Consortium improves the GO classification by importing relevant terms from external bio-ontologies into the updated version of GO that is GO-PLUS [54]. The GO-PLUS defined a new relationship which links a GO term with the corresponding term from other ontologies, mainly with three ontologies: Cell Ontology (CL) [12], the Chemical Entity of Biological Interest (ChEBI) and Uber Anatomy Ontology (Uberon) [85]. A Large fraction of these mapping are to the ChEBI ontology. ChEBI ontology provides the most comprehensive standardised and structured chemical terminology of biological interest. The GO also includes other cross-references between the GO terms and several widely-used related systems, such as Enzyme Commission (EC)[6], Kyoto Encyclopedia of Genes and Genomes (KEGG) [68] and Rhea databases [4]. However, there are a large number of GO terms that are not yet cross-referenced.

The third contribution of the GOC is the provision of a set of tools that facilitate the browsing, visualising, querying, analysis and downloading of the GO ontologies and annotations. In recent years, many types of tools have been developed with different capabilities to enable researchers to access information about the GO terms and their relationships and related gene products either manually or computationally. A large number of these tools have been designed by other biological research communities outside the GOC community [83]. Here, we provide a brief description of the current widely used tools which have been used extensively during our ontology development:

1. AmiGO [21]

[5] `http://geneontology.org/docs/annotation-contributors/`
[6] http://www.sbcs.qmul.ac.uk/iubmb/enzyme/

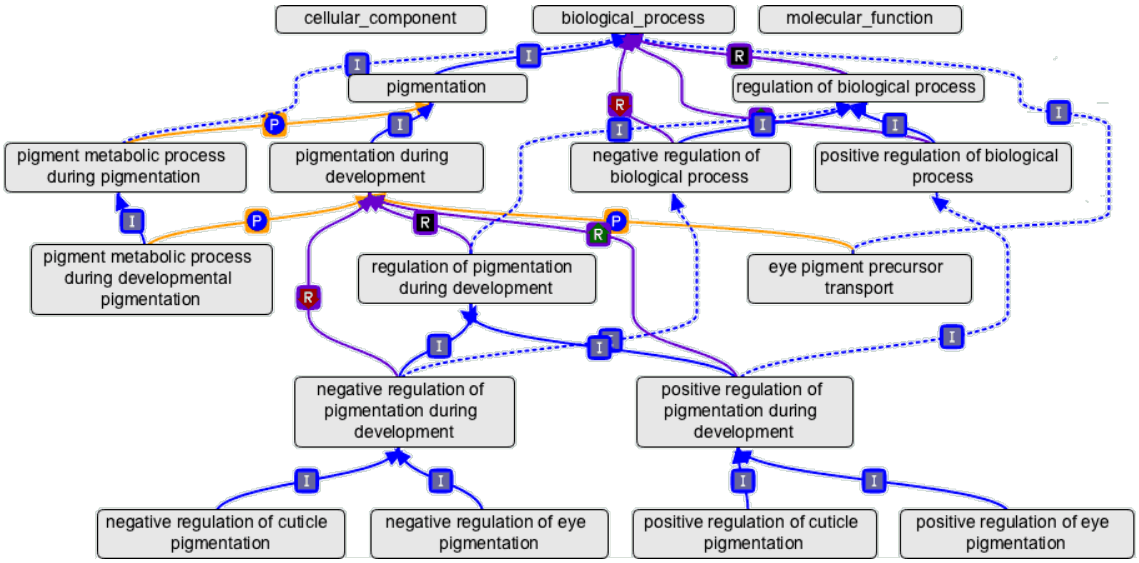


Figure 2.2: Gene Ontology sub-hierarchy, from `http://www-legacy.geneontology.org/GO.ontology.structure.shtml`

The current official web-based application developed and maintained by the GOC to allow for searching, browsing, and visualising the structure of GO and its annotations. AmiGO includes a set of features and tools:

- An interactive ontology browser: enabling GO users to navigate through the ontology structure.
- BLAST search engine: for querying against the gene products sequences annotated with the GO terms.
- Term Enrichment: to find the common functions among a set of genes using GO annotations.
- GO Slimmer: used to have a mapping between annotations of a set of genes and GO terms.
- GO Online SQL Environment (GOOSE): an interface that enables users to run queries against the GO database using SQL and download the result of query in different formats.

2. QuickGO [15]

One of the most popular web-based tools that facilitate the browsing of GO ontologies and their associated annotations. QuickGO is a product of

the European Molecular Biology Laboratory's European Bioinformatics Institute (EMBL-EBI) developed to provide the users of GO a weekly updated version of the GO ontologies allowing the search and display of information about GO terms and their relationships. In general, QuickGO includes the same information about GO terms and their relationships as AmiGO, but in a different display format. The benefit of using QuickGO is the ability to search and use multiple filtering options on both electronic and manual annotations.

In the section that follows, we will discuss several attempts aiming to enhance the Gene Ontology representation and to overcome the difficulties related to the ontology development and maintenance.

2.4.1 Previous efforts to improve GO

It has been more than two decades since the establishment of GO, meanwhile, the ontology representation has evolved in response to the development of ontological languages and techniques that facilitate ontology building and management.

An early attempt to change the informal and hand-crafted description of GO to a formal representation and logic-based description was carried out by Wroe [120] in cooperation with the GOC team. They exploited the features of the knowledge representation language DAML+OIL [60] and DL-based reasoners to provide rich, formal representation and automatic classification of GO in the Gene Ontology Next Generation (GONG) project. The use of the DAML+OIL language considerably simplified the GO development and maintenance processes.

Another effort conducted by Mungall [84] who attempted to normalise GO by transforming the textual definitions of the GO classes into computable logical definitions to be exploited by reasoners. The created logical definitions partitioned into cross-product mappings to describe the biological functions using internal and external classes from related OBO Foundry ontologies, such as proteins from PRotein Ontology (PRO) [100], chemical entities from ChEBI, CL, Sequence Ontology (SO), and using relations from the Relation Ontology (RO) [102]. For example, the MF x ChEBI cross-product class aimed to define all the molecular functions that have chemical entities as input, output or both from the ChEBI. They claimed that these

logical definitions have enhanced the GO in different aspects. Firstly, by enabling semantic interoperability between GO and other OBO ontologies. Secondly, facilitate the integration of GO data with other OBO ontologies. Finally, logical definitions can be used by reasoners which would facilitate the reconstruction and validation of the ontology automatically. However, there were no reasoner that can reasons over all the elements: GO, the cross-product set and all the referenced candidate ontologies. Alternatively, the reasoning applied on separate cross-product sets along with the associated ontologies.

In the following sections, we will investigate the most well-known methodologies and technologies that have been used for ontology engineering.

2.5 Ontology development methodologies

Ontology engineering is the set of ontology development tasks that are performed by ontology developers to construct consistent and useful ontologies. These domain-independent tasks are based on a variety of elements: methods and methodologies, principles, tools and languages which support the initiation, development and maintenance of ontologies. An ontology building methodology provides a set of guidelines and design principle which explain the decisions that need to be made in every stage of the ontology building life cycle. In the last two decades, several methodologies and best practice has been proposed either as a result of upfront design, or stemming from practical experience of a development process of multiple ontologies [61]. Although, most of the proposed methodologies were designed for the construction of ontologies from scratch, there were other approaches and methodologies developed to facilitate other development processes, such as ontology evaluation, ontology re-engineering and ontology evolution. In fact, there is no single standardised ontology design methodology that is widely used and covers all aspects of ontology development regardless of the application domain [50]. However, most methodologies share common stages: requirement specification, conceptualisation, formalisation, structure implementation and evaluation. Next, we consider the most well-known methodologies for building ontologies.

2.5.1 OntoClean

OntoClean [48] is one of the leading methodologies that was developed to ensure the correctness and consistency of the taxonomic hierarchies of an ontology. It consists of a set of formal and domain-independent meta-properties that are used for annotating the intended meaning of concepts, properties and relations defined in an ontology of any application domain. Moreover, it includes constraints on the use of the meta-properties to ensure the integrity of the created ontology taxonomy. That is, the idea of OntoClean methodology is to validate an ontology taxonomy by highlighting not appropriate and incoherent classification insights in the taxonomy so that the ontologist can correct it based on a group of general meta-properties. These meta-properties, or referred to as highly general ontological notions, adopted from philosophical ontological notions which are *identity, dependence, unity,* and *rigidity*. Later, the OntoClean meta-properties were extended by two more meta-properties that is *permanence* and *actuality* to describe the behaviour of properties in term of time and existence [117]. All entities with an ontology are associated with these meta-properties to describe the entities characteristics using three different labels for each of the meta-properties. The property *rigidity* is one of the important notions that describe how a property of an entity being essential to the instances of that entity by label it as rigid, non-rigid, or anti-rigid. For example, the property of having a brain is essential to a person, so every instance of a person must have a brain in every possible world. Another property can be labeled as a semi-rigid if there are some instances that cannot exhibit the property. Constraints can be applied on the use of these properties, such that if a property A subsumes property B and A classified as anti-rigid, then property B must be classified as anti-rigid.

2.5.2 Ontology Normalisation

The primary objective of the ontology normalisation methodology is to enable the creation of explicit and modular ontology that will overcome several issues related to ontology development via a reduction of manual maintenance, re-use of ontology classification, and facilitation of ontology evolution. The methodology is divided into two independent stages: i.e., “ontological normalisation” refers to the clean definition of the basic knowledge structure of ontology concepts using a technique

such as OntoClean. The second part is the normalisation approach to ontology implementation [93], which requires two steps: first, disentangling the structure of an ontology into two independent, disjointed taxonomies, i.e., *self-standing concepts* and *refining concepts*. The self-standing concepts refer to the core concepts of a particular domain (the physical and conceptual things in the world, e.g., "people", "animal", "phone") that may or may not be defined using other descriptive concepts and conditions. The self-standing concepts should be disjoint from their siblings, and the children need not cover their parents. The second taxonomy, the refining concepts (e.g., "small, medium, large"), are used to define self-standing concepts. In contrast to the self-standing taxonomy, the children of each refining concept should be made exhaustive (covering the parent concept), though they still need to be disjoint. Then, decomposed taxonomies are recombined based on a list of definitions, formal descriptions, axioms, and a reasoner.

By classifying ontology entities in this way, the ontologies of complex domains can be constructed from small ontologies. This mechanism allows ontologies to be constructed from several modules that can be reused, maintained, and independently developed with minimal effort [94]. As the approach relies on logical reasoners to create subsumption hierarchies by linking related modules, to allow a reasoner to build an accurate and normalised ontology, all definitions and formal description of concepts should be accurate and complete, with a clear distinction between self-standing and refining hierarchies. To this end, top-level ontologies provide basic categories and distinctions that can be used to improve the expression and accuracy of the modules in a given domain. Figure 2.3 provides an example of an ontology before and after normalisation. The original hierarchy is given on the left and the normalised skeleton taxonomies and sets of definitions and axioms can be seen on the right. Based on the lists of definitions, axioms, and restrictions, the two separated hierarchical taxonomies are recombined.

2.5.3 Ontology Hypernormalisation

A more recent approach developed as an extension of the ontology normalisation methodology is hypernormalisation [77]. In the normalisation approach, the decisions on which the different taxonomies are formed are to some degree arbitrary.

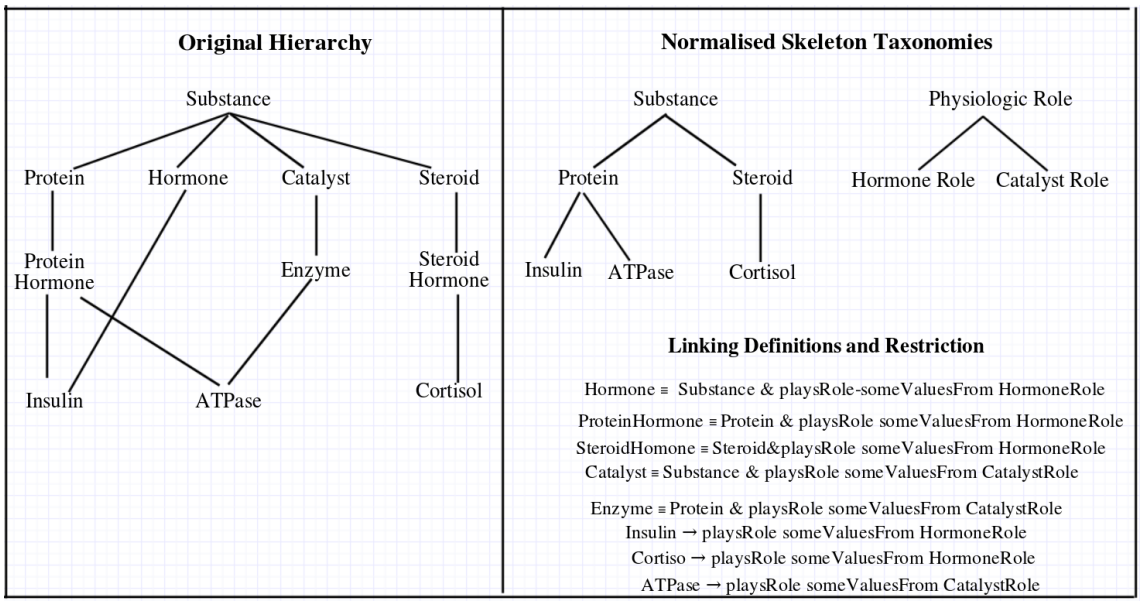


Figure 2.3: Normalised ontology of biological substances and roles, from [94]

However, this considered to be unnecessary because it can be achieved through the use of reasoning especially for the self-standing hierarchy. In this regard, the hypernormalisation technique can be distinguished from the normalisation technique in that the asserted hierarchy of an ontology is nearly or completely a flattened hierarchy among self-standing classes. Figure 2.4 shows part of the amino acid ontology [107] represented in a normalised form where the self-standing hierarchy is created, arbitrarily, from the aromaticity of the amino acids. The same ontology represented in the hypernormalised form, where the self-standing hierarchy is flattened, as show in Figure 2.5. This type of normalisation exploits logical reasoners to build the different hierarchies, instead of manually creating and editing them. For example, the concept of *small positive non-polar hydrophobic amino acid* has five parents, namely, *small amino acid*, *positive amino acid*, *non-polar amino acid*, *hydrophobic amino acid* and *amino acid*. With the right axiom patterns, the reasoner will infer the right hierarchy without the need for the developer to re-build the polyhierarchy. That is to say, the only task of the developer is to define the self-standing classes in terms of their properties, with no specific classification required. Conversely, the refining types should be correctly built.

The hypernormalisation technique is of an incremental nature, i.e., increasing the effort made during the hypernormalisation process results in greater inferred links

in the final reasoning stage. More specifically, describing every detail about the core concepts of a given domain in the refining taxonomies increases the number of relationships then inferred by the logical reasoner. As such, the hypernormalisation process depends on how much detail an ontology developer wants to model and how much they want to simplify. Moreover, hypernormalisation is not an ontology development technique that must be completely implemented, or else cannot be used at all. On the contrary, it is possible to come across some relationships that could not be captured using the hypernormalisation approach (i.e., inferred) but to be manually asserted.

Ontologies of partonomic structures are not amenable to this type of classification as their hierarchies are formed mainly using part-of relationships. For example, in anatomical ontologies, classes are structured using the two common relationships, *part-of* and *type-of* [9]. One of the main principles of creating a normalised ontology is having subsumption relationships between the primitive concepts of a domain knowledge (i.e., not a partonomy) [93], to allows a logical reasoner to infer the right classification and check the ontology's consistency. Hypernormalisation is only beneficial when ontology classes can be represented in a taxonomic tree with multiple inheritance subsumption relationships. That is to say, the purpose of hypernormalisation is to handle a complex multiple-inheritance hierarchies.

Lastly, to be able to implement this type of ontology development, OWL or other DL based formalisms are required. Within the OWL languages, OWL-DL is the sub-language of OWL required for hypernormalisation, as the process depends on the modelling constructs (e.g., owl:disjointWith) and features available with the OWL-DL. As well as this, OWL-DL provides a high degree of expression in parallel with a guarantee to maintain computational completeness. Tawny-OWL [76] (described in detail in Chapter 3) is a programmatic environment for OWL2 ontologies development, OWL2 encompasses: OWL-Lite, OWL-DL and some components of OWL-Full. In addition, Tawny-OWL provides higher-level patterns that explicitly supports the creation of a hypernormalised ontology.

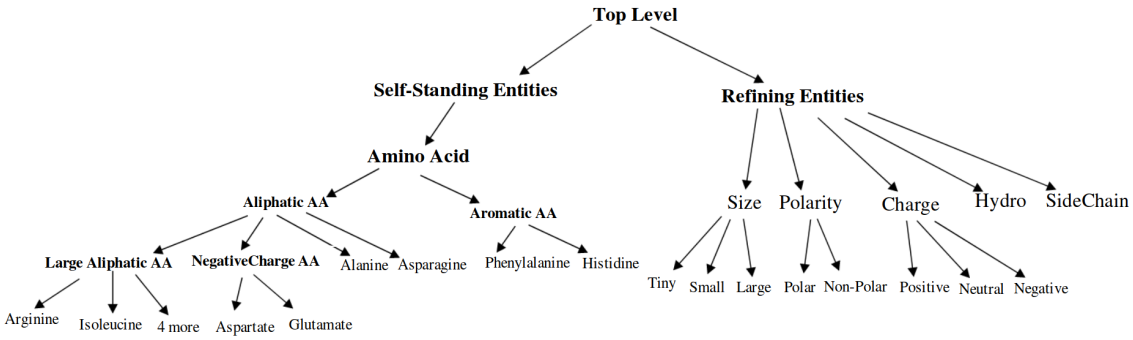


Figure 2.4: A normalised ontology for Amino Acid Ontology. Some labels have been abbreviated

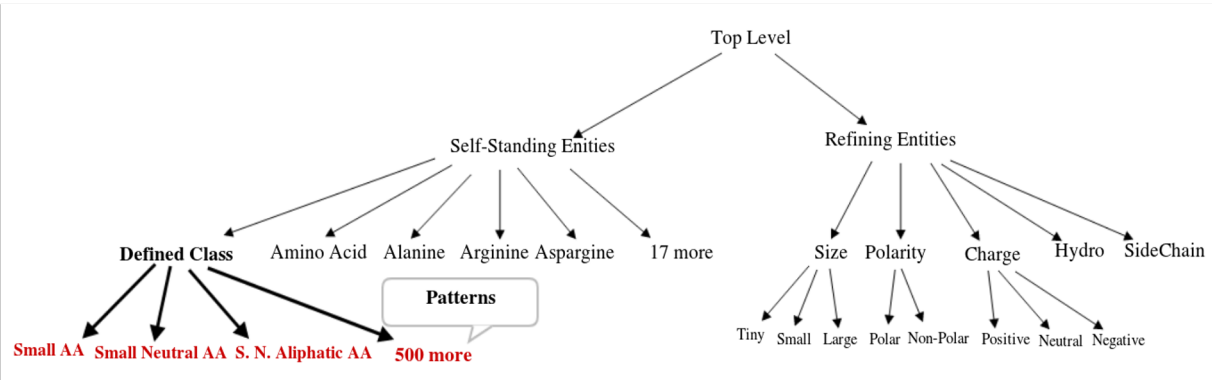


Figure 2.5: A hypernormalized ontology for amino-acid ontology slightly modified from [77]

2.6 Ontology development technologies

2.6.1 Ontology Design Patterns (ODPs)

ODPs have become a key component of ontologies development lifecycle which involves solving ontology modelling problems with acknowledged solutions. Like in software engineering, ODPs are domain-independent ontology design patterns. By applying common solutions to most recurrent problems, this allow the creation of rich and strong representation of an ontology, reduced the effort and time spent during an ontology design process and ease communication between ontology developers [42]. The most challenging and critical areas of ontology modelling is the ability to make them reusable, maintainable and easily extendable which are relatively common with complex and large ontologies. Ontology patterns can be used to support the process of ontology engineering and improving existing ontologies by encouraging the reuse of best practice and well-proven ontology code. There are several types of ODPs which have being grouped into six families; Presentation, Structural, Correspondence, Reasoning, Lexico-Syntactic, and Content ODPs [105]. A brief explanation of each ODPs are in the following:

1. Presentation OPs concern with improving ontologies readability and usability based on a user point of view. Examples of presentation OPs are Naming and Annotation OPs.

2. Structural OPs: cover Logical and Architectural ODPs.

    - Logical OPs provides solutions for problems related to ontology language inadequacy and limitations to increase the language expressivity. For example, in OWL a property is a binary relation, in order to represent a relation between more than two concepts, we need a Logical OP to enable this feature.

    - Architectural OPs specify the overall ontological structure and design internally or externally.

3. Correspondence OPs can be either Reengineering or Mapping OPs.

- Reengineering OPs consist of a set of transformation rules that can be used to generate ontologies from different source (ontological and non-ontological sources).

- Mapping OPs, as the name suggests, allow related ontologies (i.e., in a similar domain) to be linked based on some semantic relations, such as equivalence, overlap and containment.

4. Reasoning OPs aim to provide effective reasoning services related to classification, inheritance, materialisation, subsumption and de-anonymising to achieve better reasoning results on some ontologies. A reasoning pattern called Normalisation [110] is one of the well-known Reasoning OPs.

5. Lexico-Syntactic OPs (LSPs) consist of a set of syntactic structured words that can be used for extracting semantic relations (e.g., `is_a` `part_of`) from unstructured text which are then used in creating ontologies. For example the patterns of identifying hyponym/hypernym relations developed by Hearst [53] aimed to find hyponyms and hypernyms in text (see Table 2.2).

6. Content OPs (CPs) aim to address the design problems related to the content of a domain by providing solutions to define the domain classes and properties in a small ontologies (building blocks). They can be reused to solve design problems in related domains either directly or by making necessary modifications and extensions. CPs can be encoded using a representation language, however, in order to reuse them over the Semantic Web CPs need to be encoded in OWL [41].

The project of GONG proposed a number of ODPs that facilitate the migration of large biological ontologies to more formal and richer ontology language, such as OWL DL [64]. In this thesis, we investigate GO and attempt to build content patterns (CPs) that provide solutions to recurrent modelling issues that are more specific to the GO ontology.

Table 2.2: The Hearst Lexico-Syntactic OPs for finding hyponyms and hypernym relations. The NP refers to a noun phrase

No	Patterns	Example
1	NP_0 **such as** {NP_1, NP_2, NP_3..., (and or) } NP_n	Popular biological ontologies, **such as** GO and ChEBI.
2	**Such** NP_0 **as** {NP_1,} * {or and} NP_n	**Such** membrane proteins **as** Histones and C-myc play various roles . . .
3	NP_1 {,NP_N} * {,} **(and or) other** NP_0	. . . atoms, ions and molecules **or other** chemical entities . .
4	NP_0 {,} **including** {NP_1,} * {or and} NP_2	BioPortal contains a large set of bio-ontologies **including** GO and CL.
5	NP_0 {,} **especially** {NP_1,} * {or and} NP_2	..ontologies are hard to maintain **especially** large-scale ontologies

2.6.2 Ontology reasoners

An ontology reasoner is a software application that is designed to discover indirect and implicit knowledge from a set of explicit defined facts in ontologies and knowledge bases systems. The explicit facts are expressed in an ontology language, such as the OWL a description logic-based language which make the automated reasoning is possible. Reasoners plays a significant role in ontology development, as they assist in determine ontology consistency and infer the subsumption relationship between the classes described in an ontology. Many reasoners nowadays support the main set of reasoning tasks, such as ontology consistency, classification, instance checking and query answering. The GOC uses the OWL reasoners, ELK [70] and Arachne [10] reasoners. Arachne is an OWL RL reasoner, was developed to support reasoning with the GOC modelling tool Noctua Stack while developing GO "Causal Activity Models" (GO-CAM). GO-CAMs designed to show the contribution of gene products towards the implementation of biological processes.

Next, we introduce the most used ontology editors in the last few years.

2.7 Ontology editors

2.7.1 Protégé

Protégé is the most popular tool for ontology development because it provides both web-based and graphical user interface with a very rich environment. There are

more than 360.000 registered users who are using the Protégé platform to build their ontologies. Protégé developed by Stanford Medical Informatics and it is a free open-source application which allows the definition of ontology elements such as classes, properties, relationships between classes, variables, value restrictions, and many other features. One of the biggest value of Protégé is the ability to extended the core functionalities and architecture of Protégé platform with very useful features which would increase the system capabilities, developed by either Protégé official developers or contributors from all over the world.

2.7.2 ROBOT

ROBOT (OBO Tool) [62] is one of the latest ontology development tools that aims to automate ontology development tasks, with consideration of OBO conventions. ROBOT is divided into two main parts: "robot-core" that is a library consists of a set of core high-level functions that are based on low-level functionality from OWL API [58] and Apache Jena [22]. The library can be imported in any programming language runs on the Java Virtual Machine (JVM). The other part of ROBOT is "robot-command" a command-line interface that includes several different commands each of which performs a particular function some commands correspond to functions from the "robot-core" library. For example, ROBOT includes commands for running reasoner, add annotations, converting formats, module extraction and axioms filtering. Because ROBOT builds on OWLAPI, this make it compatible with several ontology syntaxes (e.g., OBO format, OWL Manchester Syntax). ROBOT is designed to be a successful substitute to previous tools, specifically OWLTools and its command-line tool the OBO Ontology Release Tool (OORT). OWLTools used by various OBO ontology projects as it provides a set of practical methods for implementing OBO-style, as well as enabling the full features of OWL API and OWL reasoner API. The conversion between OBO and OWL format is implemented using the command-line tool OORT.

2.7.3 Tawny-OWL

Tawny-OWL environment enable ontologies to be built, evaluated, tested programmatically and provides a pattern development methods. Patterns can facilitate the

development, because it eliminates the need to think about complicated subsumption relationships. Tawny-OWL allows patterns to be developed in the same place where classes and properties created, and defined as easy as classes, properties and relations being defined. Recently, a number of ontologies were developed successfully in the Tawny-OWL environment, such as the Karyotype Ontology [113] and Mitochondrial Disease Ontology [114].

2.8 Thesis terminology management

In this next section, we consider the philosophical background that helps us to understand the relationship between the world, knowledge and how we represent this knowledge. We use this discussion to develop and describe the terminology shown in Table 2.3 which we will use throughout this thesis.

The purpose of ontologies in both information sciences and philosophy fields is to represent or model things exist in reality including entities, processes, events, ideas, properties and relations with respect to their nature and structure [45]. In information science, an ontology is defined by its use, a computational artifact (i.e., both human and machine understandable) created to formally model the different categories of entities in a given domain of discourse based on the ontology creators view. However, in order to facilitate the exchange of formally modelled knowledge between software applications and users in different communities, the definitions of knowledge entities need to be expressed adequately with a clear and unambiguous meaning. One of the key situations that undermine the ability of communication and exchange of knowledge is the use of inconsistent terminology specified in the ontologies both within and outside specific communities. The absence of an agreed terminology that are used by a community to describe things exist in reality, causes a confusion about the prescribed meaning of concepts and ambiguous foundations to build ontologies from. The terms used within a community need to have the same meaning that associated with specific concepts.

Fundamentally, *words* are one of the primary ways to signify the *thoughts* in our minds about *things* that exist either as real-world entities (e.g., apple) or abstract entities (e.g., thinking). Nevertheless, it is hard to demonstrates the relationships

between the words we use (both spoken and written), the thoughts that in our mind and the things that our words and thoughts are refer to. It is because of various reasons: (a) words change over time and all time (b) a word in different situations or places means different things to different people (c) many new words added (almost daily) to language dictionaries. The formalisation of knowledge in information science has adopted the notion of "semiotic triangle" introduced by Richards and Ogden [86] to illustrate the relationships between three aspects: *concepts* which are abstract thoughts in the minds of people that refer to *objects* in real worlds that are represented using *terms, signs or symbols*(see Figure 2.6). The use of the triangle of meaning (another name for semiotic triangle) is implicit and informal and used by knowledge engineers to avoid ambiguity and justify the meaning of concepts based on their personal interpretation of either real-world objects or abstract objects.

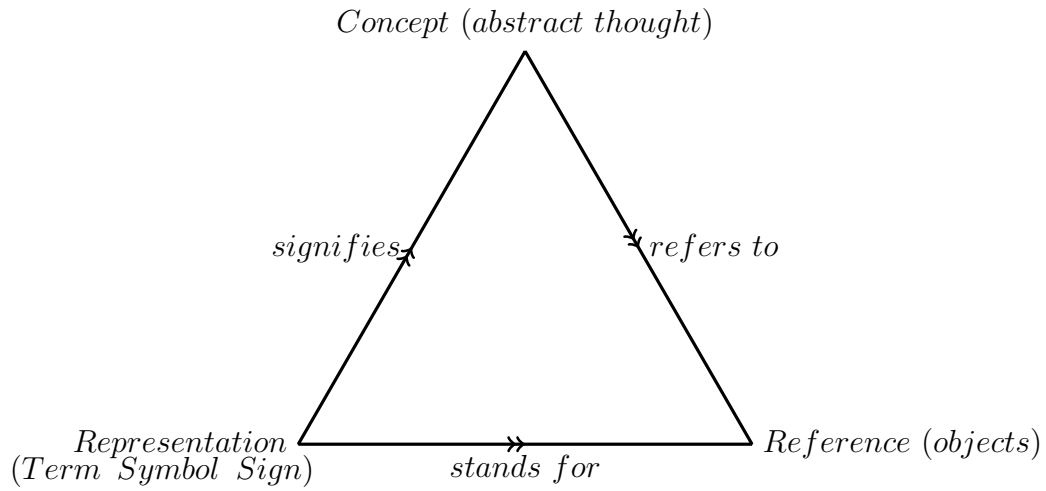


Figure 2.6: The semiotic triangle

In this thesis, we have extend the "semiotic triangle" to be specific about what the symbol is and avoid inconsistent use of terms for ontology's components and the corresponding references.

- Concept –> thought, not explicitly shared,
- Class –> symbol – decomposed into (id, label, definition)
- Referent –> individual or type

In ontology engineering, despite the different type of ontology languages, most ontologies share basic elements which are classes, instances and relations between them. Classes in an ontology represent knowledge concepts or categories. Most ontologists use the term "concept" and "class" interchangeably as synonyms, although there should be a clear distinction between the two terms. A class is a concrete representation of the underlying concept that can be defined in an ontology language using data-modelling vocabularies, such as `OWLClass`, `rdfs:label` and other class descriptions. While, a concept could be the thing that we referring to, the abstract idea. For example, two different ontologies could represent the same concept as different classes. The Gene Ontology and ChEBI have different representation of classes for same concepts. The word "term" is used inconsistently within the ontology development community: in some cases, it is used as another synonym for class or concept (the Gene Ontology uses it in this way); however, it can also be used to refer to the lexical representation of a concept, also called a label. Here, we prefer the word "label" but "term" on it's own is near synonym, term is more general – only a label when it is part of the ontology class.

In the example below we explain the issue of using same terms for different concepts and the ontological confusion that would occur because of not being precise about the meaning of concepts.

```
 * Human is label for individual member of Homo sapiens
 * Man is label for concept male human
```

Listing 2.12: Ontology 1

```
* Human is label for individual member of Homo sapiens
* Man is label for concept adult male human
* Boy is label for concept child male human
```

Listing 2.13: Ontology 2

Based on our definition:

Concept: is "Thought or reference" (same as in semiotic triangle)

Class: is Symbol, composed of "label", "identifier (url)" and logical definition.

From the statements that illustrated in both ontologies, it can be concluded that:

- Human is thought, shared by both ontologies.
- Male is thought shared by both ontologies.

- Man is label shared by both ontologies, but mapping to different, but related, thoughts or concepts, where Man in Ontology 1 is superset of Man in Ontology 2.
- Thought Man and Boy in Ontology 2 is not represented directly in Ontology 1.
- Thought Man in Ontology 2 is not represented in Ontology 1, but can be stated as "man or boy"

In our hypernormalisation ontology we are not changing the entities that exist in reality and represented by the GO classes. The GO labels we are trying to keep, we are changing the logical representation of the GO classes that we use to link together the labels to the underlying notion (the part of reality). Throughout this thesis, to overcome any confusions and to make clear distinction between a thought, or a class from specific ontology (e.g., GO and ChEBI), we use a different font style for each terminology, see Table 2.3. When referring to a class from an ontology, we include the ontology namespace (e.g., GO, MFO, ChEBI) and the class identifier (e.g., (GO:xxxxxx), (ChEBI:xxxx)).

Table 2.3: Thesis terminology

No	Terminology	definition	Font styles
1	Concept	a *concept* is the thing that we referring to, the abstract idea.	all *concepts* are written in *Italic Shape*
2	Class	a `class` is a concrete representation of the underlying *concept* that can be defined in an ontology language using data-modelling vocabularies, such as OWLClass, rdfs:label and other class descriptions.	all classes are written in `teletype-font`
3	Term	a term is the lexical representation of a *concept*.	normal font
4	Label	a *"label"* is a term only when it is part of an ontology.	written in slanted shape and between double quotes

Example, the *iron(III)* refers to the element iron in its +3 oxidation state, represented in both GO (`ferric iron binding` (GO:0008199)) and ChEBI (`iron(3+)` (CHEBI:29034)), although they use different labels – in GO is *"ferric iron"* and Chebi *"iron(3+)"*)

2.9 Summary

In this chapter, to summarise the issues related to ontology development, we first had to answer fundamental questions related to ontologies. First, we discussed the importance of ontologies in the fields of computer science and information science, and described how an ontology as a modelling technique first became widely used in the biological and medicinal domains. Second, we explained the transformation of the definition of an ontology in the context of computer science, and highlighted the difference between an ontology and other types of conceptual specification schemes. Moreover, we investigated the set of formal languages in which ontologies can be expressed, and noted the common elements that these languages offer, with which we can represent knowledge in a domain of discourse. From there, we concentrated on one of the most expressive languages, the OWL language, which has become a standardised and broadly accepted ontology language as it provides far richer and more logic-based semantics than the earlier SW language initiatives XML, RDF, and RDFS. Yet, we described how as expressive languages become very popular the computational reasoning with such languages become a challenging task.

In Section 2.2, we introduced some of the most successful ontologies, SNOMED-CT and ICD. From these, we learned about the development processes for, and challenges associated with, improving the quality and structure of these ontologies. In addition, we discussed the attempts that have been made to reformulate SNOMED-CT and ICD-11 into the OWL format, highlighting the shortcomings of these attempts.

In Section 2.3, we discussed the considerable effort that has been made by the Open Biomedical Ontologies (OBO) Foundry to address the issues associated with ontologies' development, maintenance and identification. From this, we learned about the identification policy of using monotonically increasing numbers to identify ontolog-

ical resources that are submitted to the OBO Foundry. However, these monotonic numeric identifiers, if misread or misheard, are likely to be accidentally replaced with another numbers, which can cause a race condition if two developers build a single ontology in parallel. Recognising that, we sought to contribute to overcoming the causal issue by developing a new approach to identifiers, thereby producing ones that are semantics-free, can be read by humans, and are check-summable. Hence, our first research question emerged as: "What is the advantages of this approach to ontology development?" RQ1.

In Section 2.4, we explored the Gene Ontology, one of the most successful biological ontologies. From our research, we gained an understanding of the biological knowledge represented in the ontology, and how it is structured. Moreover, we uncovered difficulties with the ontology's development and maintenance processes.

In Section 2.5, we discussed the most well-known methodologies for building and easing the development of ontologies. That led us to focus on the normalisation, hypernormalisation, and pattern-driven development methodologies. We noted how hypernormalisation has been utilised to ease the development of relatively small ontologies, which led us to raise our second research question: "Is it possible to apply this approach to a large ontology, and what we do learn from exploiting the hypernormalisation and patternisation methodologies to this end?" RQ2. We adopted the Gene Ontology as a case study with which to explore this further, and by taking this approach, raised the secondary questions: "How is the shape of the hypernormalised ontology different from the original ontology?" RQ2.1, "Do we learn anything from applying this approach"? RQ2.2, and "What type of query capability we can achieve when implementing this approach?" RQ2.3.

As mentioned in Section 2.5.3, the implementation of the hypernormalisation methodology depends on designing ontology patterns and using logical reasoners. Moreover, to be able to implement this type of ontology development methodology, it is helpful to have an ontology development environment that supports and provides higher-level patterns. Accordingly, Tawny-OWL [76] (described in Chapter 3) provides a set of patterns that explicitly supports the creation of a hypernormalised ontology.

In the next chapter, we introduce Tawny-OWL as it is the primary tool that we used to develop our Hyper-GO ontology with the assistance of higher-level patterns that explicitly support the creation of a hypernormalised ontology.

3

Tawny-OWL

Contents

3.1 Introduction

In this chapter, we introduce Tawny-OWL [76], the main ontology development tool that we have used in this research. We provide examples of how an OWL ontology and its entities can be created in the Tawny-OWL format, which has a syntax designed after the Manchester OWL Syntax [59]. The Manchester Syntax Syntax is a human-readable, frame-based syntax that has been used in various ontology development tools. Accordingly, we assert that the Tawny-OWL syntax is relatively easy to understand. In addition to this, we discuss the higher-level patterns that explicitly support the creation of a hypernormalised ontology [77]. This chapter includes no further investigation but provides additional thesis background.

Tawny-OWL[1] is a fully programmatic interactive environment for OWL ontology creation and management. It provides a rich environment where ontology-related classes, properties, and relationships between classes are created with the assistance of patterns that can be easily and accurately built. Tawny-OWL is built in the programming language Clojure (a dialect of the Lisp language) and exploits many of its programming features to move ontological development into a form of programmatic source code. Tawny-OWL utilises Clojure functions and keywords to create a frame-based syntax. Furthermore, Tawny-OWL is a fully-extensible language, whereby new syntax and features (either general or ontology-specific) can be easily added to the environment. Reasoning services are supported within Tawny-OWL via testing environments to check ontologies' consistency and enable ontologies to be queried.

Next, we will describe the core functions of Tawny-OWL, its advantages over traditional tools and emphasise the importance of higher-level patterns for building a hyper-normalised ontology. The Tawny-OWL version of the Family Ontology is used to demonstrate how OWL classes, properties and axioms are defined in Tawny-OWL. However, this work should not be considered as a user documentation of Tawny-OWL. As the Tawny-OWL repository[2] provides a comprehensive description of the tool, how to use it and the necessary tools for using Tawny-OWL. For instance, the

[1] Developed by Dr Phillip Lord, Newcastle University.
[2] https://github.com/phillord/tawny-owl

Tawny-OWL documentation recommended a user to have sufficient knowledge of OWL and ontologies in general and the Clojure language in particular.

3.2 Tawny-OWL summary

Typically, the fundamental syntax of Tawny-OWL is a Clojure expansion that mixes frame-focused Manchester Syntax with Clojure entities, meaning that an entity is governed by a Clojure function, a frame by a keyword, and finally a value by an element or expression of Clojure. Using examples from our exemplar family ontology, the expression that define the ontology in Tawny-OWL is illustrated in Listing 3.1. Any expression is a list that is (parenthesis delimited).

```
(defontology family
  :iri "http://example.com/owl/families/"
  :prefix "fam:")
```

Listing 3.1: Define family ontology in Tawny-OWL.

The entity functions labelled as "def" are functions of Tawny-OWL syntax that yield a novice symbol that enables the user to refer to the relevant OWL entities at a later point. To be more precise, the `defontology` function is developed upon the `ontology` function. The first generates the OWL API `OWLOntology` object and the second yields a symbol e.g., "family". Usually, each Clojure namespace has a maximum of one ontology determined by it, however, more ontologies may exist in some cases. Concurrent usage of the `defontology` function with the same symbol title and namespace, makes the new ontology overwrite the previous one.

In the example above, we included the ontology frames; `:iri` and `:prefix`. The first keyword frame is aimed at determining the Internationalized Resource Identifier (IRI) of the ontology. The IRI is then kept and used as a fundamental IRI for all ontological entities. The value of the frame must be entered or else, the Tawny-OWL syntax will produce a random IRI. Likewise the :prefix keyword frame is applied to determine the ontology's prefix. The value of this frame should be entered as well or else the prefix will be bound to the ontology's name. Contrary to IRI, this prefix bears no semantic value. By applying the above frames, we have determined the IRI of the ontology to `http://example.com/owl/families/` and the prefix, in this case, is "fam:". In Tawny-OWL syntax, the frames are not determined separately

but are executed by the emergence of another frame or a closing parenthesis.

You can define an OWLClass object through the `defclass` function, see Listing 3.2 for a fundamental class definition. In the example the `defclass` function merges an object of `OWLClass` with the `Human` symbol. The `defclass` function is developed based on the `owl-class` function, more specifically the `owl-class` function generates a no-name OWL API `OWLClass` object and "defclass" generates a symbol, here is `Human`.

```
(defclass Human)
```

Listing 3.2: Example of a class definition in Tawny-OWL syntax.

The `defclass` function accepts a number of frames that further describe an OWL class such as :annotation, :equivalent or :subclass. To introduce an annotation to the entity object, we use the `:annotation` frame to yield an `OWLAnnotationAxion` object, which utilises the produced `owl-comment-property` object respectively. In our case, we just introduce an annotation with the comment that describe the `Person` class (see Listing 3.3).

```
(defclass Person
  :annotation
  (annotation owl-comment-property "Represents the set of all
      people."))
```

Listing 3.3: An example of class definition with comment annotation in Tawny-OWL syntax.

In Tawny-OWL syntax, several developments apply to the same semantic and syntax references. The defined class in Listing 3.3 can be simplified and shortened through the shortcut `owl-comment`, (see Listing 3.5)

```
(defclass Person
   :annotation
   (owl-comment "Represents the set of all people."))
```

Listing 3.4: An example of class definition with `owl-comment` in Tawny-OWL syntax.

Likewise, the `label` shortcut action is applied to yield the respective `label` annotation axiom through the annotation `rdfs:label`.

```
(defclass Person
   :annotation
   (label "Person"))
```

Listing 3.5: An example of class definition with `label` frame in Tawny-OWL syntax.

Thus far, we have witnessed that the Manchester Syntax and Tawny-OWL have the same frames. For instance, the `:annotation` of Tawny-OWL corresponds to the `Annotations:` frame of Manchester Syntax. Likewise, the `:equivalent` of Tawny-OWL frame corresponds to the frame `EquivalentTo:` of Manchester Syntax (see Listing 3.6). Additionally, the frame `:disjoint` of Tawny-OWL corresponds to the frame `DisjointWith:` of Manchester Syntax. But, there are some exclusions to this general rule.

```
(defclass Person
   :equivalent Human)
```

Listing 3.6: An example of class definition with `:equivalent` frame in Tawny-OWL syntax.

The first exclusion is what has been referred to as *shortcut frames*. When developing ontologies, it is generally considered a good practice for every entity to use a simple syntax label and definition, constructed using the `rdfs:comment` annotation property and the `rdfs:label` likewise. This is supported in Tawny-OWL by offering shortcut frames to introduce and yield the most suitable `OWLAnnotationAxiom` to an entity. In Listing 3.3 to Listing 3.5, the `:annotation` frame is utilised to introduce the `label` and `comment` annotation axioms. This can be achieved using the frames `:comment` and `:label`, which are described in the class definition counterpart in Listing 3.7.

```
(defclass Person
   :label "Person"
   :comment "Represents the set of all people.")
```

Listing 3.7: An example of class definition with `:comment` and `:label` frames in Tawny-OWL syntax.

There is a second exclusion concerning `OWLSubClassOfAxioms`. In order to define super-classes in Manchester Syntax, we use the `SubClassOf:` frame while Tawny-OWL has more simpler syntax using the `:super` frame, and still have the same the logical semantic as the Manchester Syntax frame (see Listing 3.8). The reverse of the `:super` frame is the `:sub` frame, it is developed to make one or more

classes underneath the defined class. These frames can be employed to append `OWLSubClassOFAxioms` to a class object or `OWLSubPropertyofAxioms` to a property or more properties.

```
(defclass Man
   :super Person)
```

Listing 3.8: Using a `super` frame in Tawny-OWL syntax.

You can define an `ObjectProperty` through the `defoproperty` function, see Listing 3.9 for an example of an object property definition. The object property `hasWife` defined as a subproperty of the `hasSpouse` object property – this implies, when a man has a wife, he also has a spouse. We can specify constraints on the object property such that you can not use the `hasWife` property for any other class than `Man` (and its subclasses) as a domain and for any other class than `Woman` as a range.

```
(defoproperty hasWife
  :super hasSpouse
  :domain Man :range Woman)
```

Listing 3.9: An example of object property definition in Tawny-OWL syntax.

In order to explicitly capture more specific knowledge, ontology developers use universal and existential restrictions. The universal restriction is expressed through the `only` function. The simple example in Listing 3.10 defines someone as a happy person only if all their children are happy persons. One thing to note is that Tawny-OWL has a define before use semantics, define `HappyPerson` before use it. On the other hand, the existential restriction is currently the most frequently used, which is set through the function `owl-some`[3], see Listing 3.11 for an example in Tawny-OWL syntax. In the example, we define a class `Parent`, which is equivalent to someone with a least one child.

```
(defclass HappyPerson)
(refine HappyPerson
   :equivalent (only hasChild HappyPerson))
```

Listing 3.10: An example of a universal restriction definition in Tawny-OWL syntax.

[3]The use of `some` clashes with the `clojure.core`. The use of `owl-some` is longer but safer.

```
(defclass Parent
  :equivalent (owl-some hasChild Person))
```

Listing 3.11: An example of an existential restriction definition in Tawny-OWL syntax.

The OWL Boolean operators: union, intersection and complement are also defined in Tawny-OWL using the functions `and`, `or` and `not` respectively. The example in Listing 3.12 to Listing 3.14 illustrate the usage of the boolean operator functions in Tawny-OWL. The first example define a `Mother` class as a subclass of `Woman` and has to be true that every `Mother` is a parent. In the second example the `Parent` class extended using the `refine` function to also be a mother `or` father. Alternatively, this can be defined using the `as-subclasses` function with `:disjoint` and `:cover` option frames. Lastly, the `ChildlessPerson` class is defined to be equivalent to `not Parent`[4].

```
(defclass Mother
  :subclass Woman
  :equivalent (and Woman Parent)))
```

Listing 3.12: An example of using `and` function in Tawny-OWL.

```
(refine Parent
  :equivalent (or Mother Father))
```

Listing 3.13: An example of using `or` function in Tawny-OWL.

```
(defclass ChildlessPerson
  :equivalent (and Person (not Parent)))
```

Listing 3.14: An example of using `not` function in Tawny-OWL.

Defining individuals in Tawny-OWL can be achieved through the `defindividual` function, see Listing 3.15. In the example, the individual named `Jack` is defined with two types, Person and Parent, and given a fact that he `hasWife` Mary.

```
(defindividual Jack
  :type (and Person Parent)
  :fact (is hasWife Mary))
```

Listing 3.15: An example of defining an individual in Tawny-OWL syntax.

There are a number def entities provided by Tawny-OWL to enable the definition of OWL entities. Table 3.1 shows an overview of all given functions and their basic OWL API objects that a Tawny-OWL user can use to build OWL ontologies. We

[4]The use of `not` clashes with the `clojure.core`, the use of `owl-not` is longer but safer.

have given examples for most of these Tawny-OWL functions early in this section, the rest can be easily found on the Tawny-OWL project repository[5] or our exemplar family ontology repository[6].

Table 3.1: Tawny-OWL main functions, their def forms and their basis OWL API objects.

Def form	Tawny-OWL function	OWL object
`defontology`	`ontology`	`OWLOntology`
`defclass`	`owl-class`	`OWLClass`
`defindividual`	`individual`	`OWLIndividual`
`defoproperty`	`object-property`	`OWLObjectProperty`
`defaproperty`	`annotation-property`	`OWLAnnotationProperty`
`defdproperty`	`data-property`	`OWLDataProperty`

3.3 Patterns in Tawny-OWL

Tawny-OWL enables the creation of OWL ontologies in a programmatic form through a textual interface and simple and straightforward syntax. As we already showed, a simple part of an ontology can be constructed using the default, readily available Tawny-OWL syntax (using examples from the family ontology). However, a number of ontologies have complex structures and repetitive components, such as biomedical ontologies; these ontologies can be developed by adding arbitrary patterns and additional syntax, which become a standard part of the ontologies' development. Patterns can be generic, reused among ontologies, or specifically developed for a single ontology, known as content-specific patterns (CPs)(see Section 2.6.1). Like the basic Tawny syntax, most patterns are developed using Clojure functions (see Section 3.4). The first developed generic patterns are the `closure` and `covering` patterns, designed to form a Closed-World Assumption (CWA) using the `some-only` and `:cover` functions, respectively. Examples of the usage of these patterns can be found, for instance, in the definitions of the Pizza Ontology (see the ontology repository[7]). One of the main advantages of Tawny-OWL is that it allows you to develop ontological patterns and define classes and properties alongside each other in a single

[5]https://github.com/phillord/tawny-owl
[6]https://github.com/phillord/owl-primer
[7]https://github.com/phillord/tawny-pizza

file. However, to use the Tawny-OWL basic functions and patterns, a number of required namespaces such as `tawny.owl` and `tawny.pattern` must be added at the beginning of a file and before an ontology definition.

In this thesis, we will focus on a set of higher-level patterns that enable the creation of a normalised ontology. The patterns were described in detail in the hypernormalisation paper [77]. To illustrate the usage of these common patterns, we use examples from two ontologies: the amino-acids ontology [107] and the pizza ontology.

The first pattern is the *value partition*, which was developed by [90] to address the difficulties of modelling properties with continuous values by splitting the values into discrete ranges, such as defining the seven colours of the rainbow. Tawny-OWL implemented this pattern in a more simple and straightforward syntax using the `defpartition` function. The usage example of a *value partition* pattern in the amino-acids ontology is shown in Listing 3.16. Without using the `defpartition` function, the *value partition* pattern still can be implemented, but with a lot more definitions, and a relatively complex representation (see Listing 3.17). That is to say, the `defpartition` function will produce the same axioms as in Listing 3.17 but with fewer syntax. As such, the `defpartition` pattern is beneficial for building explicit refining hierarchies for large ontologies with fewer ontological definitions.

```
(defpartition Charge
  [Positive Neutral Negative]
  :domain AminoAcid)
  :super PhysioChemicalProperty)
```

Listing 3.16: An example of using `defpartition` pattern in Tawny-OWL syntax [77].

The *tier* pattern is more general pattern than the *value partition* pattern. It is designed to allow the construction of non-continues ranges with more options such as the ability to make the generated property functional or not functional and the subclasses disjointness and covering. An example usage of the *tier* pattern in the pizza ontology is shown in Listing 3.18 to define the set of toppings for a vegetable pizza. It is very unlikely to define a vegetable pizza with only one topping and to cover all the vegetable toppings in the world. To achieve this, we specify that the characteristics `functional` and `cover` to be `false` to allow for all possible combination of toppings and to be open for new toppings respectively.

```
(class Charge
     :super PhysicoChemicalProperty)
(class Charge
    :equivalent
        (or Positive Neutral Negative))
(object-property hasCharge
   :domain AminoAcid
   :range Charge
   :characteristic :functional)
(class Positive
    :super Charge
    :disjoint Neutral Negative)
(class Neutral
    :super Charge
    :disjoint Positive Negative)
(class Negative
    :super Charge
    :disjoint Neutral Positive)
```

Listing 3.17: The expanded syntax of the `defpartition` pattern represented in Listing 3.16 [77].

```
(deftier VegetableTopping
  [Mushroom Artichoke Onion Tomato]
   :domain Pizza
   :functional false
   :cover false
   :superproperty hasTopping)
```

Listing 3.18: An example of using the `deftier` pattern in the pizza ontology.

Facet is another higher-level pattern implemented in Tawny-OWL to enable the explicit association of classes and an object property. For instance, we can declare that the classes `Polar NonPolar` as facet of the object property `hasPolarity`, see Listing 3.19. Faceted classification is widely known technique has been used in library science and commercial websites to classify resources. In our example, using the `facet` pattern, a class such as `Polar` will only be used with its declared property, this would minimise the number of errors. In fact, the previous patterns (i.e., `deftier` and `defpartition`) generate new object properties that are typically restricted to the classes of the defined pattern and have the same patterns name preceded by the "has" word. That is, these patterns declare their classes as facets of their properties. Moreover, we can use the `facet` function instead of the `owl-some` function, that is, (facet NonPolar) instead of (owl-some hasPolarity NonPolar), as

shown in Listing 3.20. The `facet` function has a *broadcasts* characteristic, that is, it can take many classes and associate each class with its correct property.

```
(as-facet
  hasPolarity
  Polar NonPolar)
```

Listing 3.19: An example of using the `facet` pattern.

```
(class Alanine
        :super (facet NonPolar Neutral))

(class Arginine
         :super (facet Polar Positive))
```

Listing 3.20: An example of using the `facet` pattern.

Lastly, the `gem` pattern was developed to provide more abstract syntax of a class definition that would ease the construction of core classes of an ontology. It is built on the `facet` function, for instance, an amino acid such as `Asparagine` can be defined using the `defgem` function, which accepts a set of facets through a `facet` frame, see Listing 3.21. The `defgem` function is different from the `defclass` and `class` functions as it includes the `facet` function frame.

```
(defgem Asparagine
   :comment "An amino acid used in the biosynthesis of
      proteins"
   :facet Neutral Hydrophilic Polar Aliphatic Small)
```

Listing 3.21: An example of using the `gem` pattern.

3.4 Clojure summary

By now, we already realise that Tawny-OWL is developed upon the Clojure language. Clojure is not widespread language as C++ and Java, but is one of the top 100 coding languages, based on the TIOBE index[8]. In this section we provide only a summary of Clojure basics including fundamental Clojure functions and the principles that are necessary to comprehend the Tawny-OWL examples. For a more extensive outline of the Clojure principles and development services, please head to Clojure's official page[9].

[8]TIOBE index is an indicator of the most used programming language
[9]https://clojure.org

Rich Hickey designed Clojure to meet his requirements for a Lisp language that employed immutable data structures as its default, that had concurrency built into its design, and that had compatibility with the widely used Java Virtual Machine (JVM) platform. Clojure is a primarily functional program language that forms part of the Lisp group of languages; it has a wide range of applications and is dynamic and compiled. Amongst the many significant organisations that employers as part of their technology stacks are Walmart, Staples, and Amazon. Clojure is frequently referred to by its producers as a "functional Lisp for the JVM". As one of Lisp's modern dialects, Clojure provides support for various features:

- Individual Lisp-like syntax: the language uses a prefix syntax with parentheses being widely used.
- Clojure REPL (Read-Eval-Print-Loop) environment, which has tight integration with widely used integrated development environments (IDEs). REPL offers programmers the satisfaction of being able to obtain instant views of the outcome of the code as it is created, which encourages them to experiment and makes them more productive.
- Code as data: as with Lisp, Clojure source code is referred to as an abstract syntax tree, a valid data structure that allows for access and manipulation.
- Robust macro system that leverages code as data, which allows for metaprogramming, i.e., the ability to write code, which can then generate additional code.

The data structure allowing Clojure and fellow Lisp languages to deal with source code as data, which involves the language with its capacity for metaprogramming, are linked lists. That's why there are so many parentheses; the source code is known as an *s-expression*, a data structure made up of parenthesised lists.

Clojure employs functional programming (FP). Functions are regarded as the primary class, and by default data is immutable. If vectors, maps, lists, etc., are created, by definition they are immutable. Although the majority of functional languages, e.g., Scala and Haskell, lean in the direction of static types, Clojure favours

dynamism. With REPL, catching errors during the coding process is more simple, and the dynamism provides the code with greater flexibility and extensibility. Lastly, Clojure offer significant support for concurrency, i.e., the capacity of dealing with many tasks simultaneously, e.g. exploiting the capacities of multicore CPUs. It is easy to share immutable data structures across a multitude of threads.

In terms of Tawny-OWL, the patterns (described in Section 3.3) are created using Clojure functions with passed parameters. For Clojure, we can define functions by employing the `defn` function and we define its related parameters using a `vector` within square brackets ([]). In Listing 3.22 we can see an exemplar definition of the `max` function in Clojure, which returns the greatest of the numbers. Similarly, we can define a new function using other built-in functions, see example in Listing 3.23.

```
user=> (max 1 2 3 4 5)
5
```

Listing 3.22: An example of using the built-in `max` function.

```
(defn power
  [x n]
  (reduce * (repeat n x)))

user=> (power 4 2)
16
```

Listing 3.23: An example of defining a `power` function.

3.5 Summary

As the fundamental tool for this thesis, we here describe here the Tawny-OWL library, Clojure, and the concept of using a coding interface to develop ontologies, by concentrating on examples from the "family ontology". Moreover, we show how higher-level patterns can be used to facilitate the development of ontologies with complex structures and repetitive components. The benefits of introducing these topics set the foundation upon which a hypernormalised ontology can be constructed.

Tawny-OWL is a programmatic interface that was developed by Phillip Lord and inspired by the work on the karyotype ontology [115]. With many advantages over traditional tools, Tawny-OWL exploits the richness and robustness of a software en-

gineering environment that allows the construction of ontologies with simple syntax that non-programmers can use in a relatively easy way. More specifically, Tawny-OWL, built in the programming language Clojure, exploits many of its programming features to move ontological development into a form of programmatic source code. The library is continually evaluated and developed, this paper [77] describes the latest update of Tawny-OWL and introduces the higher-level patterns that support the creation of a hypernormalised ontology.

All the features of Tawny-OWL described in thesis are available in the 2.0 version of the library. The library itself is still in its early years, with the first release of Tawny-OWL less than a decade ago in November 2012. The majority of the code written in this thesis will be included in the Tawny-OWL syntax, and can be expected to be included unless we mention otherwise.

The implementation of the hypernormalisation technique can be achieved in a non-Tawny-OWL environment but with additional ontological definitions, effort, and complex representation compared to Tawny-OWL. For instance, performing the hypernormalisation technique in a tool like Protégé will involve a lot of clicking to create classes with related characteristics, thus requiring greater time and effort. In Listing 3.17, we showed a simplified Tawny-OWL representation of the *value partition*, and how it can be represented in a different ontological environment with many more axioms (see Listing 3.17). As examples such as that one demonstrate, applying hypernormalisation within the Tawny-OWL space makes ontology developers' work easier and less time-consuming and error-prone, especially with large ontologies.

In the next chapter, we present the work of Identitas, a new approach to identifiers that aims to improve the management of ontologies.

4

Identitas Project

Contents

4.1 Introduction

The Semantic Web (SW) builds on the W3C Resource Description Framework (RDF) [87], which is a standard framework for describing real-world concepts (e.g., physical objects or people) or representing abstract concepts (e.g., web documents). It makes use of standard web identifiers, that is the Uniform Resource Identifier (URI) [13] or, more lately, its internationalized equivalent, the IRI [36]. Entities within RDF have identifiers as a reference, which can be referred to in any linked dataset. This is a fundamental component of the SW because it enables information to be published about the resources and links to be made between different entities on the web. The URI remains a standard mechanism for identifying concepts on the SW in the semantically-extended versions of the RDF, that is, the RDF Schema (RDFS) and, recently, in the Web Ontology Language (OWL) [88], which is utilised for ontology authorship.

In the last few years, there has been a rapid increase in the number of ontologies employed to describe different scientific domains. Meanwhile, a set of standard practices has been built up; this enables improved representation, identification, and accessibility. For example, with the Open Biological and Biomedical Ontologies Foundry (OBO Foundry) [101], there is a standard usage of metadata for each ontological element, including labels, identifiers, definitions, the editorial status, and so forth. For ontologies to be reusable and accessible, the ontological components, including the ontology itself, must be able to be identified using an IRI. The IRI is a string of characters from the Universal Character Set (UCS); it can be split into global and local parts when it comes to identifying ontological resources. However, there are various incarnations of IRIs depending on the IRI scheme, such as `http`, `https`, `ftp`, `mailto`, and so forth. In this context, we concentrate on `http` and `https`, which are the most commonly used. When it comes to ontological identifiers, IRIs are normally split into two parts. The first part is the IRI protocol and authority, that is, the domain name, which is a global ID used to facilitate uniqueness the world over. The second part is then utilised for the uniqueness of the entities within an ontology. In this context, we refer to the second part as the *identifier* (for clarification, see Section 4.2.1).

This chapter is mainly about identifiers: how we generate and mint the local part of an ontology IRI, what are the rules and criteria to produce a sensible identifier to ontological entities?

Developers within the bioinformatics community recommend that identifiers are *semantics-free*, or *meaningless* [78], since an identifier that is based on some semantics associated with the term may need to be changed when that meaning changes, even if the change does not reflect a change in the ontological semantics. For instance, Apple Computers Inc became Apple to reflect the changing scope of the company. On the other hand, humans prefer it if the local part of an identifier created from natural language because then it will be *easier to memorise*, *create*, *understand*, and *pronounce*. One solution is just to hide identifiers from people, which is achievable in some environments such as Protégé. This only works, however, on the condition that we never leave these environments, which is not practical. URLs [14], for instance, have always been visible in web browsers.

Over time, different processes or schemas have been employed to coin identifiers[1]. In this chapter, we consider the advantages and disadvantages of these, and introduce Identitas, a library that coins identifiers in such a way that it overcomes some of the disadvantages we have witnessed. The software is available from `https://github.com/Nizal-Shammry/identitas-j`. It has been integrated into environments for ontology development such as Tawny-OWL (See Chapter 3) and Protégé (see Section 2.7.3).

4.2 Background and related work

4.2.1 IRI syntax for ontology

Ontologies, databases, and their components can be globally identified using IRIs; this extends the exiting URI scheme, which is limited to the ASCII set of characters. As such, every URI or URL is an IRI, but not vice-versa. Though there are many IRI protocols, `http` and `https` have become the standard protocols employed to

[1]Unfortunately, the use of terms like "identifier" and "local ID" is used so inconsistently across the field, that we are unlikely to avoid confusion. In this chapter, we have defined the term "identifier" carefully and distinguished it from "IRI", and parts of an IRI such as "fragment"

identify things on the Semantic Web. The generic syntax of a `http` IRI consists of several components; those are the protocol, authority, path, query, and fragment. An example of an ontological IRI, is shown below; it consists of a compact sequence of characters that can be divided into two main parts, the global and local. The purpose of the first part, on the left, is to provide access to a resource on the web using a combination of the protocol, authority, and a path. In this example, the scheme is *https*, the authority is *purl.uniprot.org*, and the path is *uniprot*. Then, the global part is followed by a fragment. This last element of an IRI, the local identifier, is unique within a database or an ontology. It can either can be generated automatically (e.g., accession number such as `0000008` or an alphanumeric such as `A0A022YWF9`) or can be typed manually as a lexical name. In this case, the local identifier identifies a subordinate resource, which is mostly defined in or is a part of a primary resource (e.g., an ontological document or a database). The local identifier is separated from the rest of the IRI by `(#)` or `(/)` characters; in this case, it follows the last `(/)` character. In the example `P08100` identifies a resource, a type of protein, that is defined in the UniProt Knowledge Base. The IRI does not need to contain a local identifier; in the case, it refers to the primary resource (i.e., the whole ontology).

The first, global section of an IRI (its base) is likely to be common across all entities in a given ontology and requires data providers to interact with external bodies to serve the ontology globally, which is mostly the authority. On the other hand, the local identifier, which is mostly included in the last part of the syntax, is created based on the preferences of a specific ontological creator.

4.2.2 *Existing identifiers schemes*

In this section, we consider a number of common identifier schemes, and we use this to pull out the general characteristics[2] of these schemes, as are identified in Table 4.1.

IDs schema	*semantic − free*	*speakable*	*meaningful to humans*	*global*	*unique*	*stability*	*persistent*	*interoperability*	*resolvability*	*Fee − based*
OBO	Yes	Yes	No	No	Yes	Yes	Yes	Yes	No	No
OBO (PURL)	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No
LEI	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes
DOI	Yes	No	No	No	Yes	Yes	Yes	Yes	No	Yes
DOI (doi.org)	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
LSID	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	No
LSRN	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No
DCMI (PURL)	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
UUID	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	No
ISBN	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	No

Table 4.1: The list of desirable characteristics provided with identification schemas

The OBO Foundry is an intensive effort to provide shared, logical, well-formed, and scientifically-valid knowledge in the biological and biomedical domains. Participants in the OBO Foundry are defining a set of principles and methodologies to facilitate interoperability of ontologies and enable data integration. There are over 60 ontologies that have been retrofitted or built on the basis of the principles of the OBO Foundry, such as the Gene Ontology (GO) [43] and the Chemical Entities of Biological Interest (ChEBI) [52]. An ID policy has been established and is recommended for ontologies that are submitted to the OBO Foundry. The syntax of OBO Foundry IDs is constructed of a base URI, followed by an IDspace (also known as a namespace) and a local identifier. The IDspaces, associated with a URI as a prefix, are unique to the OBO Foundry (e.g., GO, CHEBI, CL) and are followed by monotonically increasing numbers (e.g. `00000001`). GO followed the OBO Foundry policy to identify its terms with a unique seven-digit identifier prefixed by GO, for instance, `http://purl.obolibrary.org/obo/GO_0000016`. In addition to this, all ontologies hosted in the OBO Foundry library have a Persistent Uniform Resource Locator (PURL), which always resolves to the OWL or OBO version of the ontologies. PURLs are URLs utilised to permanently identify resources on the Internet by redirecting the PURL link, using a resolution service, to reach the URL target. The OBO Foundry policy illustrates that they are strongly committed to the *semantics-free* identifier approach, dictating that their local identifier should not

[2]The words in italics

consist of labels or mnemonics that are *meaningful* to humans; a meaningless identifier with a meaningful label *(provided through rdfs:label)* is employed to describe the characteristics of the entity.

A Legal Entity Identifier (LEI)[3] is an alphanumeric global identifier developed by the International Organisation for Standardisation (ISO) to identify the legal entities participating in financial transactions. The LEI identifier consists of 20 characters, starting with four numeric characters referring to the LEI-issuing organisation, known as a Local Operating Unit (LOU), followed by 14 alphanumeric characters that are unique to a particular LOU. In addition, two checksum digits are added for the purpose of detecting errors. As a result of the Global Financial Crisis (GFC) in 2008, governments and private sector parties have come to support the development of a Global LEI System (GLEIS) that provides an open, unique, and persistent identifier for each transaction. Currently, all financial institutions worldwide that are involved in financial transactions are required to have an LEI identifier.

Another ISO standard identifier system is the Digital Object Identifier (DOI) [89], which was approved in 2010. So far over 100 million DOI identifiers that have been assigned a DOI, mostly academic publications. The DOI syntax consists of two main parts separated by a forward slash: a prefix, which identifies the naming authority (such as DataCite or CrossRef) and a suffix, the local part, that denotes a unique DOI for a specific data object; this can be chosen by a registrar and is unique to a given prefix. The prefix consists of two components, the "Directory" and "Registrant" codes, separated by a full stop (period), for instance, `10.1000`, where the authority is the DOI Foundation itself. The (local) suffix, on the other hand, has no limitation on its length and can be a numeric or alphanumeric string. For example, in following DOI, `https://doi.org/10.1038/nphys1170`, `10.1038` denotes the registration agency (CrossRef) that manages the assignment of DOIs, `nphys1170` is the *meaningless* object identifier, and `https://doi.org/` is the *resolver* [66]. For each DOI assigned to an object to make it *(resolvable)* on the Internet, there must be metadata and a URL that automatically redirects a user to a useful online in-

[3] `https://www.gleif.org/en`

formation about that ID. As is stated on the official DOI website[4], any DOI used to define a digital object will never change, that is, *(persistent)*, even if the data associated with the ID or the physical location has changed.

The Universal Unique Identifier (UUID) [72] is another widely employed identification system, which generates a sequence of 36 characters (32 alphanumeric characters and 4 hyphens) with no centralised authority using an algorithm, in such a way that there is a low probability of another object having the same identifier. The most common version of UUID consists of three combined components; a network address of the UUID-generating host, a timestamp and a set of random alphanumeric characters. An example of a UUID is `C68EB119-46D5-46D0-B79D-77C2D897C19F`. However, the UUIDs are not globally *resolvable*, which means a resolution service is required to make the information about an entity available on the web.

The same issue of *resolvability* occurs with the Life Science Identifiers (LSID)[5] that are used to describe biological data. The LSID syntax consists of a URN label, schema name, authority identifier, namespace, and entity identifier. It is not directly *resolvable* unless a web service provides a specific location for a resource; For example, *urn:lsid:ubio.org:namebank:11815* [66]. On the contrary, a Life Science Record Name (LSRN)[6] offers entry into a centralised repository of resources from existed databases and provides a physical location (URL) for each resource within a specific database, such as PubMed, GO, or INSD (GenBank, EMBL, DDBJ); For example, `http://lsrn.org/PMID:18077722`.

Obviously, the OBO Foundry, LEI, DOI, UUID, and LSID identification systems do not try to make their IDs *meaningful* to humans; the main purpose of their identifiers is the *stability* of their IDs. To enable humans to understand and use them, there is additional metadata, such as that offered by the Dublin Core Metadata Initiative (DCMI) [116], which promotes *interoperability* and global semantic understanding by providing a set of elements that can be used to improve ontologies' meaningful names. The DCMI offers a set of metadata as a standard across different domains with fifteen vocabulary terms such as *Title, Publisher, Subject*, and so forth.

[4] `https://www.doi.org/index.html`
[5] `http://www.lsid.info`
[6] `http://lsrn.org/`

However, there are ontologies and similar resources that have been developed using *meaningful* names as part of the IRI fragment. The *meaningful* part is normally encoded in a common lexical encoding format choosing from *Single word, CamelCaseStyle, Underscore style, or Hyphen-style* or a combination of the last three styles [80]. For example, if we consider a class with this IRI, `http://www.co-ode.org/ontologies/pizza/pizza.owl#VegetarianPizza` from the pizza ontology[7] the class name *VegetarianPizza* is the *human-readable* identifier. This can have several desirable effects: people can remember such classes easily and the need for additional descriptive information is avoided; as a result, some ontologies with *meaningful* identifiers do not have labels. However, there is no doubt that there are some drawbacks associated with meaningful identifiers. For instance, they can be misspelled or mistyped and there is inherent *instability* as the meaning associated with the identifier may change over time.

One final possibility would be to use a semi-readable identifier; although not common in ontology development, these are widely seen elsewhere: for example, UniProt has in their terminology "identifiers" that supplement "accession numbers"; so P08100 is also known as OPSD_HUMAN, which derives from the name "Rhodopsin" and species. Another example is `What3words`, which uses three words to identify geographical areas that would otherwise need numeric longitude and latitudes [65].

4.3 Motivation

There has been a lot of discussion about the best practices for using and styling identifiers to identify ontology resources. The perceived wisdom is that identifiers should be semantics-free or meaningless [78]. Semantics-free identifiers have their advantages but there are a number of distinct disadvantages too, especially for humans. They are, for instance, poorly mnemonic, hard to differentiate from each other, and relatively difficult to read. For this reason, many bioinformatics databases provide both semantics-free *accession numbers*, which are essentially the same thing as an identifier in ontology terminology, and an *identifier*, which is rather like a compressed, syntactically predictable label. For example, UUID is a useful identifi-

[7] Developed by the University of Manchester

cation scheme but not beneficial in situations where identifiers need to be readable, spell-able, memorable or entered manually by hand [49][25]. The motivation behind this work is to provide a new approach to semantics-free identifiers that keeps the power of meaningless identifiers, maintains the capacity of humans to read them, reduces the chance of error, and enables concurrent development of ontologies.

4.4 Identitas

Identitas library provides a new approach to identifiers that has the potential to improve the management of ontologies and overcome some related issues with monotonic, numeric identifiers, while remaining semantics-free. We describe our solutions, and present the Identitas library, which implements the following features: concurrent development, pronounceability and checks for errors.

4.4.1 Concurrent Development

Some ontological identifiers, such as those used by OBO Foundry, increase monotonically. This causes a significant race condition if two developers build a single ontology in parallel. If both attempt to add a new term, they must both *coin* a new identifier, which must be unique. This is impossible to achieve without some degree of coordination. One typical strategy is for developers to pre-coordinate using pre-allocation schema. For example, one developer would be allocated the IDs from 1 to 1,000, another would get 1,000 to 2,000, and so on.

This approach is effective but requires developers to manage the IDspace accurately, as well as reducing the overall IDspace since preallocated IDs cannot be used elsewhere. Another approach is just-in-time coordination; for example, the URIGen [37] server enables this approach in some projects, such as the Experimental Factor Ontology (EFO) and Software Ontology (SWO), allowing them to manage their namespaces. It is based on the use of a centralised server to manage the creation of identifiers. However, this requires the developers to set up a connection to the URIGen server, therefore, it is sensitive to network issues or a lack of availability of the URIGen server. A final approach is to use temporary IDs, and then allocate final IDs at a single, coordinated point in the development process; URIGen also supports this approach and uses it to enable offline development.

We propose a much simpler approach, which is to simply use random IDs not just as temporary identifiers. While randomness does not *a priori* completely remove the potential race condition, given a large enough identifier space, the chances of collision can be reduced to provide world (or universe) uniqueness. This approach is commonly used with random Universal Unique Identifiers (UUIDs), mentioned in section 4.2.2, being perhaps the most common example. Therefore, using random IDs while developing an ontology allows the removal of the requirement for co-ordination either live, with temporary identifiers, or with pre-allocation of blocks.

4.4.2 Pronounceability

The use of randomness raises a secondary issue. These identifiers are likely to be relatively long, exacerbating the problems around memorability and pronounceability. One solution to this problem is not to show the identifiers to humans. With tools like Protégé, this is possible, of course, because it has a view, which may be different from the underlying model. With text file-formats, including an OBO format, the various OWL serialisations, and the Tawny-OWL [76] programmatic representation, this is rather harder (although the latter does provide a mechanism for achieving this). This is also difficult for the programmers developing tools like Protégé itself, who are themselves using general tools such as IDEs, debuggers, and version control systems.

We have considered using a dictionary-based approach, to replace numeric identifiers with English words. However, this approach increases the probability of selecting a word that is inappropriate or unfortunate, such as the Sonic Hedgehog gene mutations that cause holoprosencephaly in humans. Instead, we are investigating a solution in the form of *proquints* [119]. Taking this approach, a library is built to encode numbers as a set of strings of alternating consonants and vowels. Each consonant provides four bits of information and each vowel only two bits, as shown in Figure 4.1. Thus, sixteen bits can be represented using five letters, that is, three consonants and two vowels.

For example, a numeric identifier 10 associated with some term in a given ontology would be translated using the *proquint* function to `babab-babap`, while 1000 would be translated to `babab-bazom`, which is a fairly readable, spell-able and pronounce-

```
Four-bits as a consonant:
-------------------------

0 1 2 3 4 5 6 7 8 9 A B C D E F
b d f g h j k l m n p r s t v z

Two-bits as a vowel:
--------------------

0 1 2 3
a i o u
```

Figure 4.1: Encoding a 16-bit string *proquint* of alternating consonants and vowels

able string. In practice, if used to represent random numbers, *proquints* would rarely be so close in alphabetic space. It should be noted that *proquints* map directly to a single number, so they can be freely converted in either direction, and order alphabetically in the same way as the numbers that they map to.

As an extension of the original algorithm, we also provided conversions from the Java short and long data types, which provides for either a larger identifier space or less typing. Table 4.2 demonstrates the conversions from Java short, integer, and long random numbers to the equivalent strings of alternating consonants and vowels.

Short-Integer-Long numb	*ProShort*	*ProInt*	*ProLong*
0	`babab`	`babab-babab`	`babab-babab-babab-babab`
1	`babad`	`babab-babad`	`babab-babab-babab-babad`
2	`babaf`	`babab-babaf`	`babab-babab-babab-babaf`
3	`babag`	`babab-babag`	`babab-babab-babab-babag`
MIN_VALUE	`mabab`	`mabab-babab`	`mabab-babab-babab-babab`
MAX_VALUE	`luzuz`	`luzuz-zuzuz`	`luzuz-zuzuz-zuzuz-zuzuz`

Table 4.2: The conversions from Java short, integer and long random numbers to ProShort, ProInt, and ProLong strings

We must note that the short-range, at 2^{16} numbers, is large enough for most ontologies that are currently in operation. However, it is far too small when combined with randomness as, due to the birthday problem, it is likely to result in collisions even for small ontologies [111]. The long-range, meanwhile, at 2^{64} numbers, is likely to cope with all ontological applications where the identifiers are allocated as a result of human action; it has half the bit-length of a UUID, which has a 2^{128} range.

4.4.3 *Error checking*

We would like to note here that the monotonic numeric identifiers suffer from a final problem. As well as not being mnemonic, if a numeric ID is misunderstood, then it is very likely that the incorrect ID is still actually a valid one; for instance, `GO:1903424` and `GO:1904324` are IDs that differ by one number. A solution to this problem is well-understood with the use of a checksum. The use of a checksum digit allows straightforward error detection and it is an industry standard for ensuring numbers are actively transcribed. For the Identitas library, we employ the Damm algorithm [81]. This algorithm is designed to operate on numbers, but it will work on *proquints* too, as they can be converted to numbers. Examples of valid or invalid random numbers are shown in Table 4.3.

Random number	*ValidationStatus*
327890	valid
328790	invalid
328792	valid
327892	invalid

Table 4.3: Example of random IDs and their validity

4.5 Results

The result of this effort is a new style of identifiers dedicated to the notion of semantics-free IDs, and equipped with a set of desirable characteristics. The Identitas IDs, which can be read by humans, are formed from a set of characters with alternating constants and vowels, which can be generated randomly; this facilitates the process of concurrently building and developing ontologies. Furthermore, errors between similar identifiers are detected in this style using the Damm algorithm [81]. We provide an implementation of these features that can be freely combined. All of the features are implemented in our library, Identitas-j, which is developed in Java and is available in Clojure as well. We have integrated the style into two ontology development environments, Protégé and Tawny-OWL [76], and might later provide an implementations for other ontology development environments. This form of identifier space has the potential to improve the management of ontologies at little

cost.

A preliminary result of Identitas has been presented and accepted as an abstract paper in our work [6]. This work is extended by the integration of Identitas into two existing ontology development environments. Figure 4.2 is a screenshot from the Protégé interface where Identitas, particularly the random ProLong IDs, is employed in the Protégé software environment, while Figure 4.3 presents a simple example of a class created and identified with an Identitas ID. In addition, we have built and distributed a Maven [82] artifact for Identitas[8], which is available for users who want to utilise Identitas as a library in their own project.

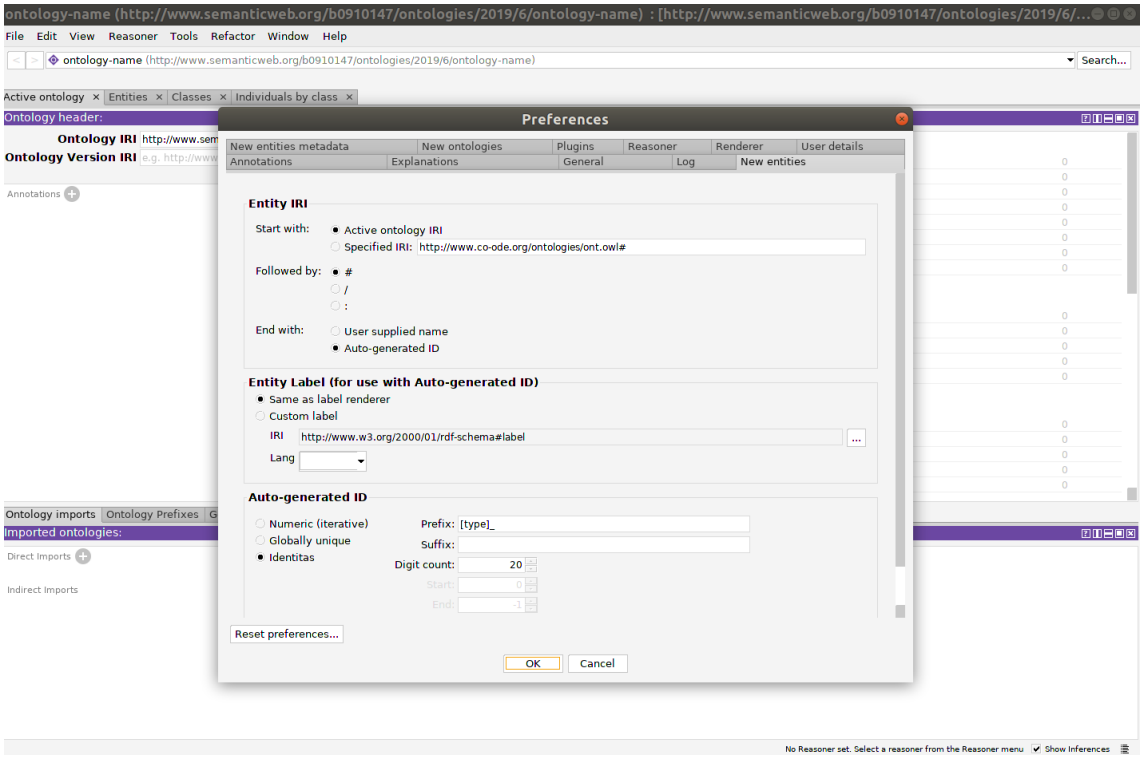


Figure 4.2: Identitas as part of the Protégé environment

4.6 Evaluation

4.6.1 Let's port GO! How easy would it be?

GO is one of the most successful ontologies, due to its consistent description of gene products across databases. Currently, it consists of more than 45,000 classes [118] describing the different biological functions of gene products over three sub-ontologies.

[8]https://mvnrepository.com/artifact/uk.org.russet/identitas-j

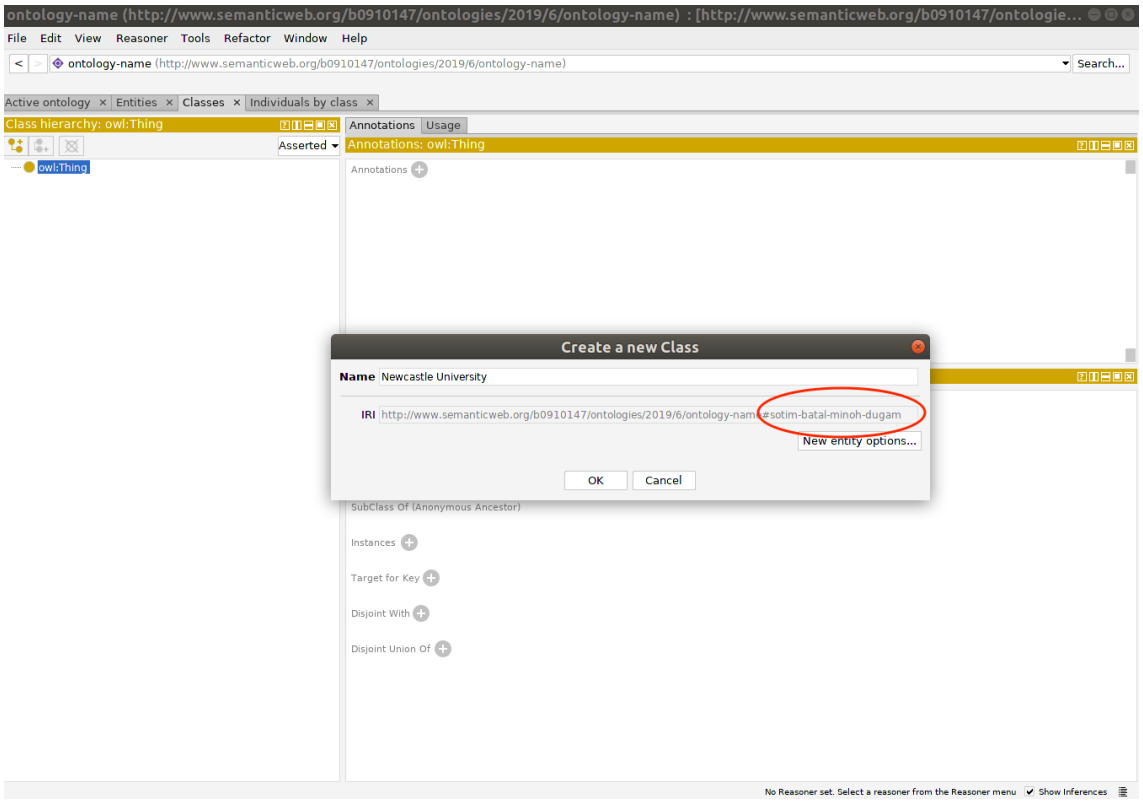


Figure 4.3: Example of a class (Newcastle University) created in Protégé and identified with identitas ID

GO adopted the OBO Foundry ID format (i.e `IDSpace:accession_number`, explained in 4.2.2) as it is designed to be easier to read for humans than other formats. Every term within GO has a name that can be read by humans (e.g., transcytosis) and a semantically meaningless GO ID (seven unique, increasing numbers prefixed by GO, e.g., GO:0045056). These characteristics make it useful as an exemplar when we consider the applicability of Identitas; would it be possible in principle to port GO to use Identitas?

From the ground up, GO could be built using our technology, taking into account the current size and growth rate of the ontology over the past ten years. GO has been growing linearly as it involved a certain number of people during its development process, as can be seen Figure 4.4. The size of GO grows by 4.39 percent each year, which gives us a potential total number of terms of around 70,000 terms after ten years and over three million terms 100 years from now. The graph in Figure 4.5 illustrates the probability of a collision when using the *proInt* (i.e., 2^{32}) and *proLong* (i.e., 2^{64}) spaces. As the size of the ontology increases, the risk of generating a

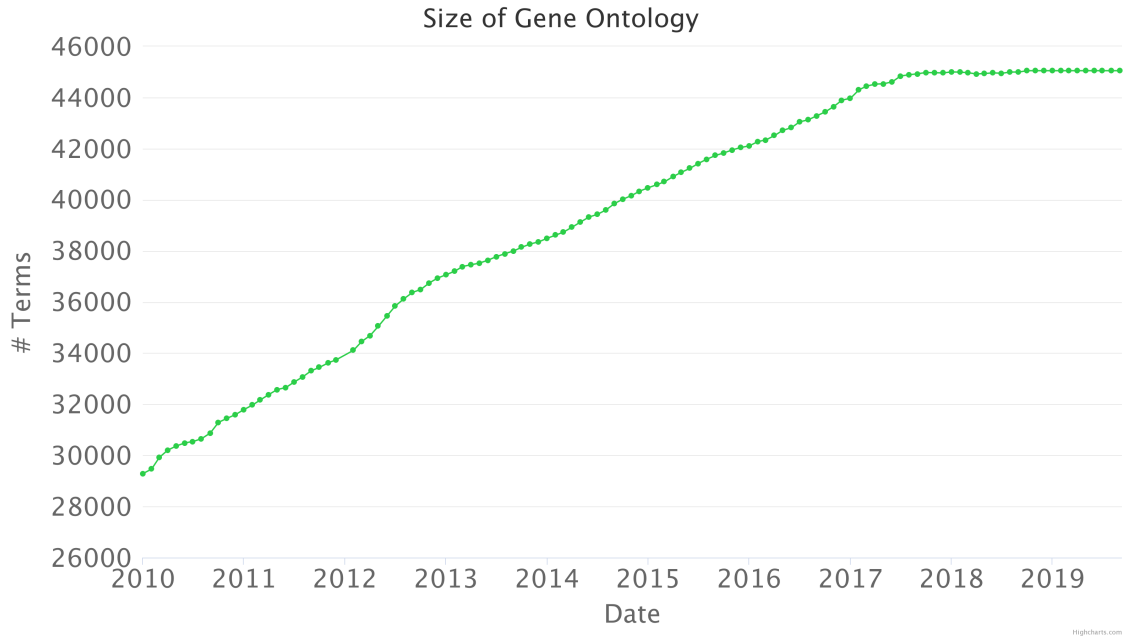


Figure 4.4: The size of GO over the last 10 years, from [63]

random duplicate ID and assigning it to different terms goes up. The number of random *proInt* IDs to be generated for a 50 percent chance of collision without the error-correction scheme is roughly 78,000 IDs. That is expected to be the number of terms after nearly 35 years after the establishment of GO. Using the checksum scheme with *proInt* will reduce the space and increase the probability of collisions at an early stage.

However, by using the random allocation technique, GO will be secure from generating an ID more than once. The *proLong* space will be broad enough to maintain both the checksum and randomness schemes, not only for GO, but for all ontologies. In practice, it is entirely acceptable to start using the *proInt* and, when the probability becomes higher and irritating, *proLong* can be used thereafter with no need to change the *proInt* IDs that have already been given. The chance of creating one duplicate ID, with the current size of GO and using *proLong* with an error detection scheme, is 10^{-10}; without the error-detection scheme, it is 10^{-11}. When the number of GO terms exceeds three million sometime in the next hundred years, on the other hand, the probability of a collision including the checksum is 10^{-6}. Although, the *proLong* might be harder to use, it scales to all of the BioPortal ontologies and tax-

onomies[9] with a probability of a clash of 10^{-5}. Because the probabilities of collisions for random *proLong* IDs are considerably small, they can hardly be seen on the chart in Figure 4.5; therefore, they are illustrated on a different graph in Figure 4.6.

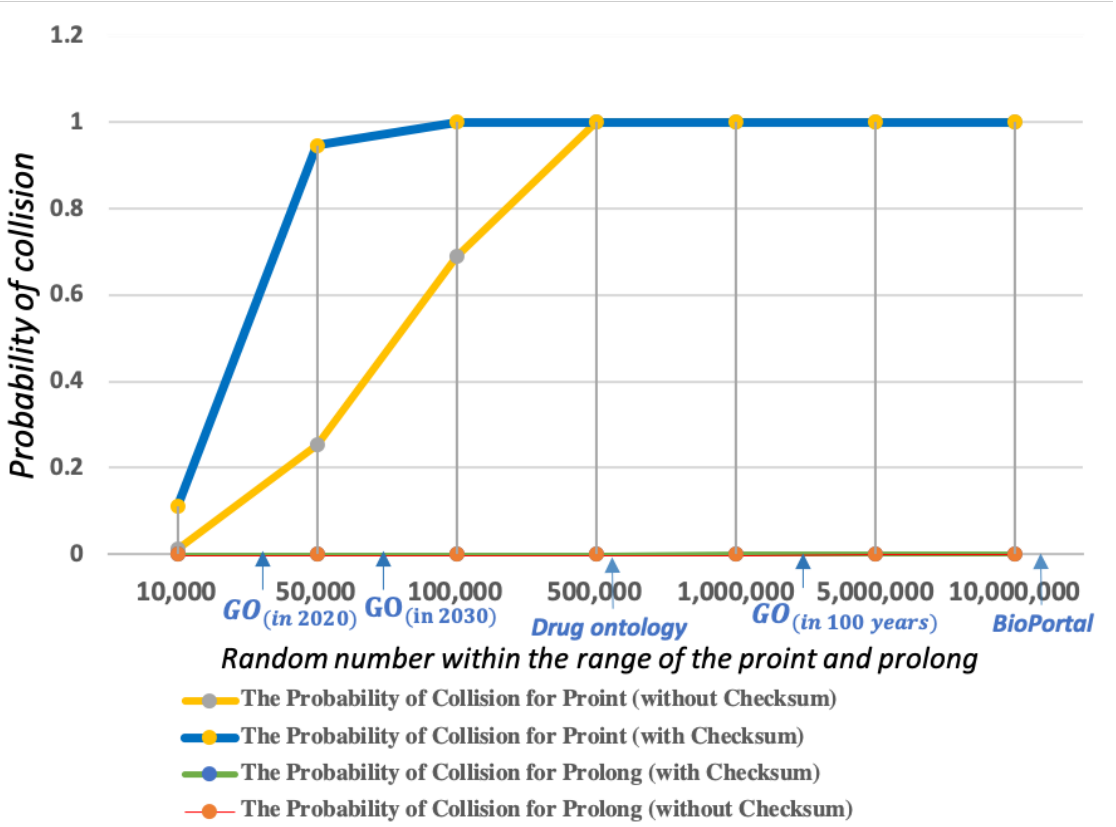


Figure 4.5: The computed probability that at least one collision occurs versus random numbers within the range of proInt 2^{32} and proLong 2^{64}

Another suggestion is to use our scheme to define GO concepts for the current state in a non-destructive way. This is possible by leaving GO IDs as they stand now, in the OBO Foundry style, and identifying any new term with an ID from the Identitas library. That is, we keep the old style for GO IDs until a term becomes obsolete, and then define the replacement term with our scheme, without changing the IDspace as it becomes intensively populated. However, before deploying the Identitas identifiers, we need to decide which *proquint* is acceptable (i.e., *proShort, proInt or proLong*). In the case of using the random allocation schema and error correction features, *proLong* identifiers are more acceptable in the long-term. On the other hand, *proShort* and *proInt* spaces are relatively small for covering all of the

[9]The world's most comprehensive repository of biomedical ontologies, having 821 ontologies and more than 10 million classes

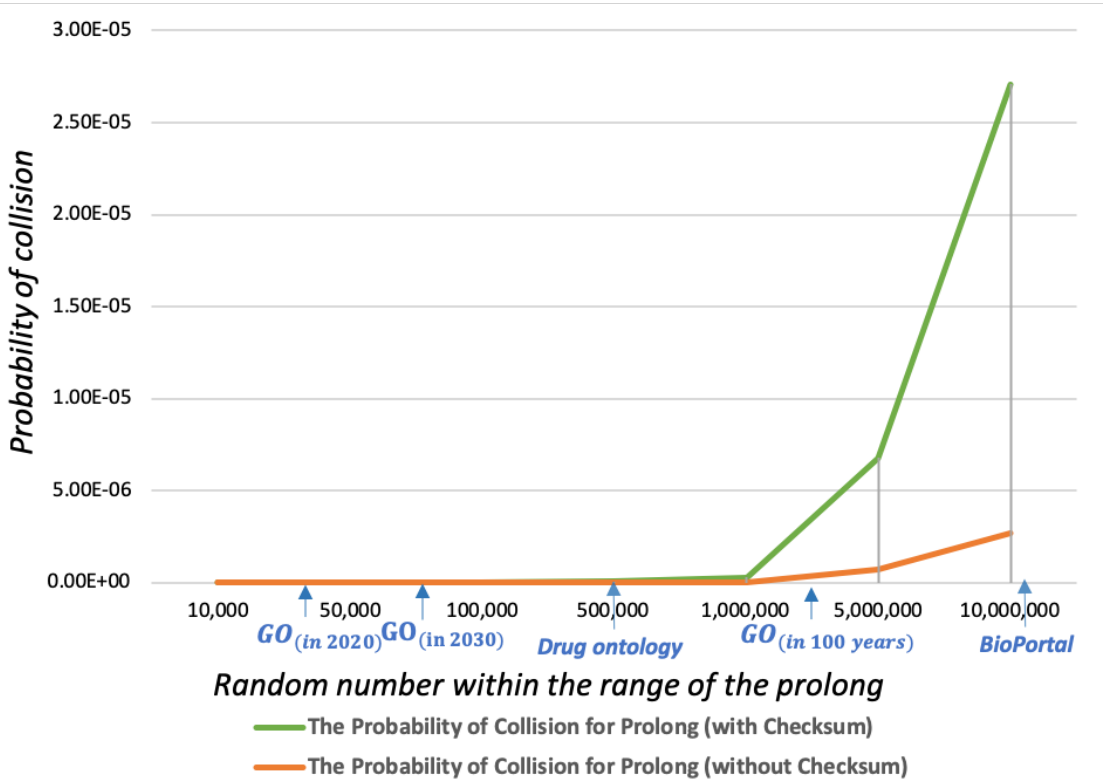


Figure 4.6: The computed probability that at least one collision occurs versus random numbers within the range of 2^{64}

GO terms and maintaining randomness and the checksum schemes simultaneously. However, there are various ontologies where *proShort and proInt*, including checksum and randomness features, would be large enough to cover their classes.

4.7 Discussion

URIs and their various incarnations (i.e., IRIs and URLs) are our way of identifying data records and resources on the web and providing a standard mechanism for their access. This allows ontologies and their elements to be reusable and shared across different domains. However, while the IRI recommendation provides a syntax, it does not specify how to use all the parts of that syntax. As a result, there are several common identification schemes, such as OBO, UUID, DOI, LEI, and LSID, each of which has its own format for local identifiers (i.e., the latter, localised part of an IRI) with different characteristics and limitations. Table 4.1 provides a summary of those identification schemes and the key characteristics that define them. An identification scheme should meet basic requirements:that is, to be unique, stable,

persistent and resolvable [66]. Most identification schemas provide these desirable characteristics, either directly by themselves or using supported systems. However, the style of the identifiers generated by most schemas either consists of numeric or alphanumeric characters, which introduces a number of problems for end-users. These can be, for instance, hard to read, memorise, and pronounce, and it is easy to make mistakes. Although these issues can be resolved with meaningful IDs, there are serious drawbacks associated with them. For a start, meaningful IDs can be spelled incorrectly or mistyped, and they are unstable as the meaning related to an ID may change over time. Another challenge with some identification systems is that require a degree of coordination between ontology developers to obtain new IDs; this can hamper or delay development work significantly. Lastly, monotonically increasing numbers, if misread or misheard, are likely to be accidentally replaced with another number that is also a valid ID.

In our project, we have implemented a new scheme for generating local identifiers, which enables the development of identifiers that are semantics-free and can be read by humans. We have avoided the issues associated with employing sequential or semi-sequential numbers by utilising random numbers and a checksum technique respectively. This will increase developers' productivity, reduce the need for coordination, and increase the power that we have to detect errors, especially when IDs are transmitted by people, a practice that is common for scientific ontologies.

In this contribution, we have presented details of the scalability of Identitas, showing that it can easily scale to an ontology the size of GO. We have, in fact, added Identitas to a refactored version of GO that we have written as part of our research – in this case Identitas IDs were added as a standard annotation property, effectively adding a secondary identifier to the main IRI. While it is clearly possible therefore, to do this, we note that we do not address the question of whether it is practical; the form of GO IDs is assumed by many pieces of software, and porting GO IDs would be a significant effort.

Although we use proquints in this scheme to avoid use of numeric IDs, neither we nor the original author have tested their readability through users studies; this would be valuable work to do, but is outside the scope of the current research. We do note

that Identitas is composable; it would be possible to add a different pronounceability layer perhaps akin to WhatThreeWords should proquints prove problematic.

We believe that the Identitas scheme is a solution that represents a good compromise between data providers who prefer IDs constructed from natural language (with identifiers that are meaningful to humans) and the need that we have for meaningless IDs, which need never be changed. In this paper, we have demonstrated that the scheme is applicable and scales easily to befit the size of current ontologies. We have also considered ways in which current ontological practices could be migrated towards the use of this scheme. We have implemented a library that generates these IDs and integrated them into existing ontology development tools. Obviously, any change has a significant barrier to adoption in a pre-existing ontology; we offer Identitas as a possibility for the future.

4.8 Summary

In this contribution, we present a new approach to identifiers, one that aims to improve the management of ontologies. This approach overcomes some of the main flaws associated with the existing approaches, providing alternative solutions. Ontology identifiers are the key for each entity defined in an ontology, and enable a unique and persistent reference to each term. The form of identifiers has been the subject of discussion, which has resulted in a number of different schemes. It is often recommended that identifiers for ontology terms should be semantics-free or meaningless. One practice, is to use numeric identifiers, starting at one and working upwards. However, this has a number of disadvantages: it does not allow for concurrent development; is relatively hard to read; and it is difficult to detect errors when an identifier is misused. From the perspective of ontology development solving these issues could significantly facilitate the process of building and managing ontologies. Here, we suggest random identifiers to enable concurrent development, while exploiting the *proquint* library to overcome the problems of memorability and pronounceability. Finally, a checksum is implemented to prevent the occurrence of errors while accessing relatively similar identifiers.

In the next chapter, we will investigate the Molecular Function Ontology (MFO)

structure and the possibility of rebuilding the ontology using the hypernormalisation technique. patterns that explicitly support the creation of a hypernormalised ontology.

5

GO Molecular Function Ontology (MFO) Analysis

Contents

5.1 Introduction

The Gene Ontology (GO) project provides a common, structured and controlled vocabulary to describe three aspects of gene and gene products in living organisms. It consists of three non-overlapping sub-ontologies, namely Biological Process Ontology (BPO), Cellular Component Ontology (CCO) and MFO (see Section 2.4). Each of the three independent ontologies covers a key area of the biological domain. Briefly, the classes of the BPO represent the biological targets for multiple molecular activities performed by a gene. The cellular component classes refer to the locations in a cell where the molecular activities take place. Lastly, the molecular function classes defines the set of activities carried out by individual gene products. For example, the MFO `binding activity (GO:0005488)` is a broad molecular functional class and includes a set of more specific functional classes, such as `bent DNA binding`, which represent narrower function, class information shown in Definition(1)

Definition: 1. *a molecular function by which a gene product interacts with DNA in a bent conformation* [1].
bent DNA binding (GO:0003681)

These three ontologies are large, comprehensive and are used by many downstream databases describing biology. However, the size and scale of the Gene Ontology is associated with a number of issues: GO is difficult to develop, re-use and maintain. Currently, it consists of more than 45,000 classes [118] describing the different functions of gene products distributed over three sub-ontologies. Table 5.1 shows the GO ontologies with the total number of classes and instances relationships that each ontology has, according to the GO Online SQL Environment (GOOSE) (see Section 2.4). The reputation and importance of the ontology and the realisation of issues have motivated restructuring initiatives. As a result, several efforts highlighted the GO issues and offered their suggestions to improve the GO representation addressed in section 2.4.1, starting from the most notably the Gene Ontology Next Generation (GONG) project [120].

In this thesis, we investigate the usage of the hypernormalisation, patternisation and programmatic approaches by asking how we could use this approach to rebuild the

[1] We use different font style for a concept, a class and label, see Section 2.8

Gene Ontology, specifically the MFO. Hypernormalisation and patternisation are techniques, which support managing the development and maintenance of ontologies. The hypernormalisation (explained in details in Section 2.5.3) process can be summarised into two main steps: disintegrate the current skeleton of the ontology into independent classification, *self-standing classes* and *refining classes*, and rely on set of patterns and automated reasoner tool to build the polyhierarchical classification. However, due to the complexity and richness of the knowledge represented in the ontology, preliminary steps need be carried out. First, we need to investigate and study the hierarchy, biological and ontological nature, logical structure, and number of classes and relations defined within the MFO.

Fundamentally, the representation of gene functions in GO is determined in the following manner [33]. This allows us to understand the biological nature of the MFO.

- A gene (also called functional unit) is a demarcated sequence of Deoxyribonucleic Acid (DNA), which includes instructions for the cell to create large macromolecule or several macromolecules.
- Gene products are the macromolecules that have been produced by the cell based on gene instructions. **Proteins** are the most common functional macromolecules responsible for performing functions for the cell, which occur at the molecular level. In GO, the functions performed by different proteins are called **activities, (e.g., catalysis or binding)**.
- The interaction between (two or more) macromolecules from non-identical genes result in a macromolecular complexes. Many essential cellular functions are performed by macromolecular complexes.
- The actions or activities of a gene product or complex are represented in MFO as **classes**. That is, the MFO **classes** describe the activities instead of the molecular entities (i.e., molecules, gene products or complexes), however, these activities take place through physical interactions with the entities.
- Although a gene is the source of the actions, gene products performs the **molecular activities** in a specific location relative to the cell.

GO ontologies	GO ID	Classes	Relationships
Molecular Function (MF)	GO:0003674	10781	14039
Biological Processes (BP)	GO:0008150	29384	71372
Cellular Components (CC)	GO:0005575	4044	7854
Total		44209	93265

Table 5.1: The number of classes and instances relationships recently comprising GO, as of February 2018.

5.2 MFO overall analysis

Based on an initial analysis, the MFO is formed from 15 top-level classes. Each of these classes describe a broad molecular activity, constructed in a hierarchy of related activities using the fundamental *is_a* relationship and segregated from other broad molecular activities. For example, the MFO binding activity (GO:0005488) is hierarchically isolated from other activities, such as `antioxidant activity (GO:0016209)`, `transporter activity (GO:0005215)` and `catalytic activity (GO:0003824)`. (Excepted are the `antioxidant` and `catalytic` activities, which have many classes that belong to both categories). Therefore, we believe that the hypernormalisation of the MFO would be achieved through hypernormalising its high-level classes separately as a starting point. Although the MFO high-level classes share attributes, they describe different tasks such as transporting chemical entities, transmitting signals, combining molecules and converting one entity into another. We start investigating the high-level classes independently to understand their biological characteristics and ontological design. Then, we apply the hypernormalisation mechanism by disentangling the structure of the selected high-level class into disjoint taxonomies: the *self-standing classes* and classifications of *refining classes*. An overview of the workflow used for hypernormalising the MFO is shown in Figure 5.1.

The ontology of molecular function refers mainly to activities that occur at a molecular level as a result of the functions performed by genes or gene complexes, such as *transport*, *catalyse*, *regulate*, *modulate* and *bind*. Within the MFO the top-level classes `catalytic activity`, `binding activity` and `transporter activity` are

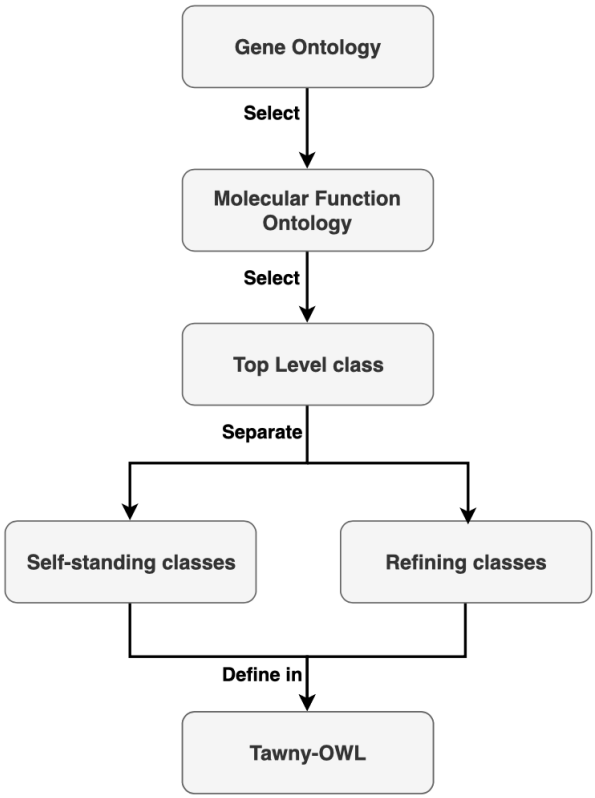


Figure 5.1: Overview of the MFO hypernormalisation workflow.

the broadest, together constituting approximately 94% of all classes in the MFO. `Binding activity` signifies the linking between ligands (signalling molecules) and receptors (either intracellular or cell surface receptors). `Transporter activity` signifies the movement of substrates such as macromolecules, small molecules and ions between cells, within a cell, and into or out of a cellular component. `Catalytic activity` (known as enzyme activity) signifies the catalysis of biochemical reactions at physiological temperatures using common types of catalysts such as enzymes. The MFO high-level classes are presented in Table 5.2, including the total number of classes each top-level class has. They are descriptive classes (i.e., they do not represent a specific activity) but include large number of sub-classes, which are more specific in representing molecular activities that typically require a physical interaction with biological/chemical entities to perform their actions. The molecular activities described through the MFO classes, like all GO classes, are defined with a human-readable name (i.e., a label), GO ID, textual description of the potential activity and relationships to other classes that represent related activities.

For example, consider the class `light transducer activity, (GO:0031993)` and its definition in Definition(2), which is classified as a *subclass-of* the general classes `energy transducer activity`.

Definition: 2. *absorbing of energy from one or more photons and transferring their energy to another molecule, usually a protein, within the cell.*
light transducer activity (GO:0031993)

Each of the MFO classes is structured into a hierarchy, using only a `is_a` relationship to classes represent broad activities and often more specific activities. Moreover, a molecular activity can be `part_of` other activities that are classified as biological processes from BPO or `occurs_in` a specified location in some cellular components form CCO where the function is active. Not only that, recently there has been a collaborative effort to link GO classes to corresponding classes and resources from external ontologies and classification systems, such as Chemical Entities of Biological Interest (ChEBI) ontology [54], Cell Ontology (CL) and Rhea (Annotated Reactions Database and Enzyme nomenclature database). This has resulted in the creation of `cross-references` and `cross-Ontology` relations. For instance, the chemical in the MFO class `carbohydrate derivative transmembrane transporter activity (GO:1901505)` is cross-referenced to the ChEBI `carbohydrate derivative (CHEBI:63299)` and the biochemical reaction defined within the `catalytic activity 2-aminoadipate transaminase activity (GO:0047536)` is mapped to the RHEA entry `(RHEA:12601)`. However, the mapping is made manually and there are a large number of GO classes have not yet been cross-referenced.

We documented all information about GO: statistics, issues and solutions and comments on GO classes, and illogical classification and modelling differences between GO and related ontologies during the project development; all available at `https://github.com/phillord/hyper-go/blob/master/nizal_notes.org`.

In the next section, we investigate further into the structure of some of the MFO top-level classes and describe our methodology for hypernormalising their hierarchies.

GO ID	GO MF Top-Level classes	No.descendants	Percentage
GO:0016209	antioxidant activity	25	0.2%
GO:0005488	binding	1856	17.2%
GO:0038024	cargo receptor activity	12	0.1%
GO:0003824	catalytic activity	7001	66.4 %
GO:0140104	molecular carrier activity	13	0.1%
GO:0098772	molecular function regulator	201	1.8 %
GO:0060089	molecular transducer activity	413	3.8%
GO:0045735	nutrient reservoir activity	0	0.0%
GO:0044183	protein folding chaperone	0	0.0%
GO:0031386	protein tag	0	0.0%
GO:0005198	structural molecule activity	41	0.3%
GO:0090729	toxin activity	0	0.0%
GO:0140110	transcription regulator activity	94	0.8%
GO:0045182	translation regulator activity	7	0.06%
GO:0005215	transporter activity	1118	10.3 %
	Total	10781	100%

Table 5.2: The top-level classes of the MFO, as of February 2018.

5.3 Top-level classes Hypernormalisation

As mentioned above, we intend to hypernormalise the hierarchy of the molecular function ontology via the individual hypernormalisation of each ontology high-level class, because the top-ranked MFO-activities (i.e., the activities that have the largest number of classes (see Table 5.2)) are largely hierarchically segregated. Figure 5.2 shows part of the hierarchical classifications for the high-level classes. During this stage we describe the procedures we follow towards producing a hypernormalised ontology (**Hyper-GO**). Firstly, we begin by developing our biological knowledge about the selected activity using GO documentation, references and related articles. Secondly, we study the logical representation of the defined notions, and more importantly analyse the textual descriptions associated with each class, because definitions illustrate the classes' molecular functions and include reference(s) to the source of the information. Then, as a result of the first step and further inspection we should identify the scope of the analysed activity, and recognise the ontological nature of the entities (i.e., small molecules, proteins or cellular components) associated mainly with the molecular function we investigate. In the last step, we

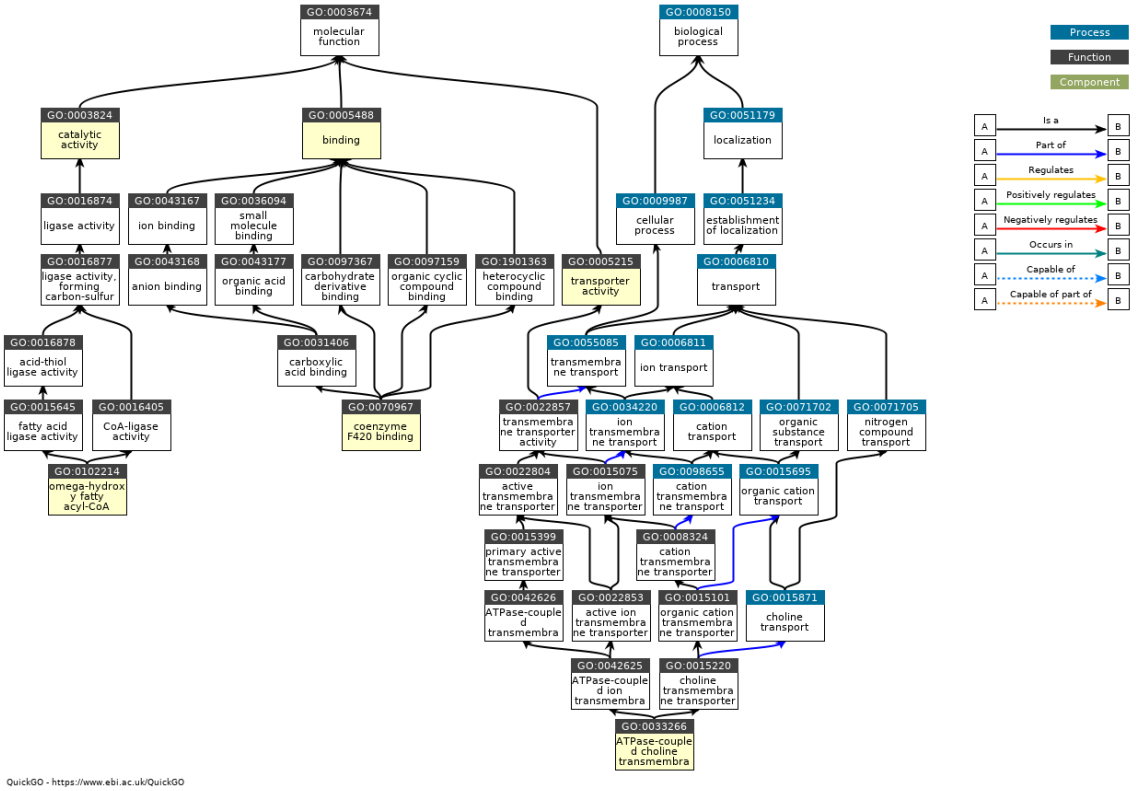


Figure 5.2: Part of the top-level MFO classes hierarchical classifications.

restructure the hierarchy of the top-level activity into disjoint trees: *self-standing classes* and *refining classes*, with the assistance of a set of designed patterns and logical reasoners to build the different hierarchies. We use part of the current state of the GO hierarchy to demonstrate how we can disentangle the ontology structure into two disjoint taxonomies: *self-standing classes* and *refining classes*.

The process of top-level class hypernormalisation can be divided into several stages, as described in the following subsections.

5.3.1 Develop biological knowledge

An essential step towards refactoring large ontologies such as GO is acquiring sufficient knowledge in each area comprising the ontology. GO has been developed by a small group of expert biologists to include large-scale biomedical and biological information, which has become comprehensive over the course of 20 years. Therefore, in the early stage of our research, it was crucial to strengthen our knowledge using internal and external resources such as the Gene Ontology Handbook [33], official online websites (`http://geneontology.org`) and related publications. This

would allows us to understand the current classification of the MFO activities and, eventually, to be able to classify the ontology classes into the two hypernormalised hierarchies. Moreover, by gaining sufficient knowledge about that specific activity, we would avoid any existing illogical classification and improve the overall representation with accurate information. Each high-level class related to the molecular function ontology demonstrates a biochemical activity in a broad sense and includes various distinct biological and chemical aspects, such as chemical entities, chemical reactions, enzymes, concentration gradient, ligands, voltage-gated channels and so on. Within the `Catalytic Activity (GO:0003824)` hierarchy, the set of enzymes classes, such as *hydrolases* and *ligases*, play key roles in catalysing biochemical reactions. Without knowing the enzymes' categories, their biological functions, types of reaction participants (reactants and products) and reaction directionality, the development of the catalytic activity class would be difficult and error-prone. The study of the field not only improves our knowledge but also reflects on the overall representation of the classes in the **Hyper-GO** ontology.

The outcome of this effort is:

- Gaining enough biological knowledge in the area of the selected MFO high-level class.
- Identify all the different terminologies used in the ontology and their biological functions.
- Having an overall picture of possible patterns in the domain of interest.

5.3.2 Classes textual definitions analysis

In GO, the class name (or using the terminology described earlier, the *label*) are human-readable but only provide a brief description of the molecular activity that a class represent. Frequently, a class label includes the type of an activity e.g., *binding* and the participant molecule e.g., *histone* that engaged in that specific activity in a biochemical context. Additionally, there is much knowledge expressed within the MFO class definition that is not possible to capture without analysing the class textual descriptions. For instance, consider the GO class GO:0015126 in Definition(3).

Definition: 3. *The directed movement of bile acid and bile salts out of a hepatocyte and into the bile canaliculus by means of an agent such as a transporter or pore.*
canalicular bile acid transmembrane transporter activity (GO:0015126)

This specifies the starting location *hepatocyte* of the molecule *bile salts* and the destination *canaliculus*. Therefore, in order to discover the different biological aspects, we must apply text analysis on the entire class's textual descriptions within a specific high-level class hierarchy. Based on an initial analysis, we found many classes have almost identical definitions, only differing in the molecular entities. This is because MFO uses standard definitions for each of the top-level activities and their sub-activities (see Listings 5.1 to 5.5). Understanding the standard definition for a high-level activity only allows us to identify the general functional patterns, whereas more sub-patterns exist with narrower grouping classes that require further analysis. Moreover, there is nothing in the class name or definition to indicate their relationship with other classes. In a number of cases, class definitions are opaque and meaningless, which necessitates an additional effort to clarify the molecular functions associated with these classes. As in the case of the classes that describe a specific type of *transporter*, such as *active transporter*, definitions specify the type of energy by which molecules are moved against their concentration gradient. In our work, the aim is to have computable definitions for all classes in GO, especially the definitions that include semantic relationships to external ontological entities, such as cellular components and chemical entities.

The outcome of this analysis is:

- Recognise the distinct biological properties in order for them to be defined in the refining skeleton as descriptors of the core classes.
- Discover the set of patterns (i.e., both general and specific patterns), mostly from the class textual descriptions.

```
Interacting selectively and non-covalently with a X, (brief
description of the activity and X]).
```

Listing 5.1: Binding activity classes standard definition.

```
Catalysis of the reaction:
[reactants = products], (reaction catalysed
```

```
  by a specific enzyme)
```

Listing 5.2: Catalytic activity classes standard definition.

```
Combining with X, to initiate a change in cell activity.
(brief description of X]).
```

Listing 5.3: Template for **X** Receptor activity (Large part of Molecular transducer activity).

```
Enables the transfer of X, into, out of or within a cell,
or between cells.
```

Listing 5.4: Transporter activity classes standard definition.

```
Modulates the activity of a X, [brief description of X].
```

Listing 5.5: Molecular function regulator activity classes standard definition.

5.3.3 Identify broad categories within a high-level class

In MFO, there are two elements specifying the molecular function represented by a class: the class's definition and its relationships to other classes of broad classifications. This is because the GO classes are represented in a directed acyclic graph (DAG), which means any class with some molecular function will also inherit all the functions that its parents have and appear in the DAG [33]. During this stage, we review any subsumption hierarchy that includes a large number of classes with a general description within the hierarchy of a specific high-level class. This allows us to understand the scope to which high-level classes limits their subclasses. For instance, most of the classes created within the top-level class of `molecular transducer activity (GO:0060089)` are subsumed under the grouping class `signaling receptor activity`, see its definition in Definition(4)

Definition: 4. *combining with molecules to receive and transmit signals from and to different places within a cell to initiate a change in cell activity.*
signaling receptor activity (GO:0038023)

The purpose of this review is:

- To identify any biological qualities, which have not being captured with previous steps.
- To understand the depth of knowledge represented in that particular high-level class.

5.3.4 Identify the ontological nature of classes

The next step is to rebuild the ontology structure with relevant and consistent representations of entities. One of the main objectives of the Open Biomedical Ontologies (OBO) Foundry is to encourage the reuse of ontology classes that others have already represented, in order to increase orthogonality between these ontologies [44]. Our analysis shows that a large number of the GO classes have references to other candidate OBO ontologies and related classification systems, including ChEBI ontology, CL, KEGG, Rhea and the Enzyme Nomenclature Database. Within MFO the molecular activities involve physical interaction with chemical entities, with some playing chemical, application or biological roles. For example:

biotin is an organic heterobicyclic compound that has the biological role of *B vitamin*, which plays important roles in *cell metabolism*.

ligand is any molecule or ion that capable of chemically binding to a cellular protein called *receptor* to initiate a change in cell activity.

amiloride is known for its application role as *antikaliuretic-diuretic agent*.

GO-PLUS [54] is considered to be the most expressive edition of GO, because it links the GO classes with equivalent classes from domain-related ontologies, mostly the ChEBI classes. This has resulted in the creation of `cross-references` and `cross-Ontology Relations` relations. ChEBI is a reference for chemical entities, specifically small chemical compounds classified according to their molecular structure, base role within a biological environment and as subatomic particles. However, the work of GO-PLUS is still incomplete and there are many classes in GO that have not yet been cross-referenced, for several reasons.

1. The alignment between GO and ChEBI was based on string-matching and manual revision.
2. GO and ChEBI ontologies use different labels for same concepts.
3. Some GO classes represent general grouping classes (e.g., *basic amino acids* and *sugar*), while ChEBI is more specific in defining classes (e.g., *lysine*, *arginine*, *histidine* and *monosaccharide*.

4. Outside the scope of the ChEBI (e.g., *lactoferrin* and *transferrin*),
5. Not yet defined in ChEBI, missing chemical entities need to be submitted to the ChEBI web application.

Our aim is to integrate the ChEBI structural hierarchy into our development environment and reference any chemical entities within the MFO classes to their corresponding chemical entities in the ChEBI ontology. Then, we aim to use an automated reasoner to classify the created classes based on their ChEBI classification, infer additional relationships, and check for any inconsistencies in the generated hierarchies. It is possible to build our ontology without using ChEBI; in this case, we will need to create an ontology of chemical entities. However, this requires enormous effort, experience, and time, which prevent us from progressing to applying the hypernormalisation approach to GO. By relying on high-quality ChEBI classifications of chemical terminologies, we overcome the challenge of dealing with the complexity of the biological relations between the chemical entities.

However, our analysis shows that there are some areas of semantic disagreement between the GO and ChEBI ontologies. One significant difference between GO and ChEBI is in the representation of acids and their conjugate bases. An acid becomes a conjugate base when loosing a proton, while a base turns into conjugate acid by accepting a proton. ChEBI uses the pair of relationships *is conjugate acid of* and *is conjugate base of* to link acid molecules with their conjugate bases and vice versa. Conversely, GO does not distinguish between the acid and the base, which from a chemical perspective, is wrong. The reason for this lack of distinction is probably that GO is describing a biological situation, where all the chemicals in question are in solution where this distinction is less meaningful. This has led to inconsistent representations of chemical entities in some GO classes' names, their definitions and their cross-ontology relations. For instance, consider the definition(5) of the GO class `L-ascorbic acid transmembrane transporter activity, GO:0015229`, where the class name refers to the acid **L-ascorbic acid** and the class definition, exact synonym and cross-ontology relation refer to the base form **L-ascorbate**.

Definition: 5. *Enables the transfer of* ***L-ascorbate*** *from one side of a membrane to the other.*

Class name: L-ascorbic acid transmembrane transporter activity (GO:0015229)
Exact synonym: L-ascorbate transporter activity
Exact synonym: vitamin C transporter activity
Cross-Ontology Relations: CHEBI:38290, L-ascorbate

The *L-ascorbate* is not the same as *L-ascorbic acid*, but is a conjugate base of *L-ascorbic acid*; in ChEBI this distinctions is represented explicitly as `L-ascorbate (CHEBI:38290)` *is conjugate base of* `L-ascorbic acid (CHEBI:29073)`, which in turn *is conjugate acid of* the `L-ascorbate (CHEBI:38290)`.

In GO the `L-ascorbic acid transmembrane transporter activity (GO:0015229)` has the relationship *is_a* to the following classes as its direct superclasses[2]:

1. `vitamin transmembrane transporter activity (GO:0090482)`]
2. `carboxylic acid transmembrane transporter activity(GO:0046943)`
3. `monosaccharide transmembrane transporter activity (GO:0015145)`
4. `organic anion transmembrane transporter activity (GO:0008514)`

This classification did not match with specific *L-ascorb** molecule, but to the union of the *L-ascorbic acid* and *L-ascorbate*. One solution discussed before [54] would be to expand these classes into large disjunctions, through the use of general concept inclusions (GCIs). So any transporter transporting GO `L-ascorbic acid` would be transporting ChEBI (`L-ascorbic acid` or `L-ascorbate` or `D-ascorbic acid` etc). This makes the axiom space larger when reasoning, and is also not necessarily correct – for example, transporters probably either have the capacity or in practice transport the D- Form, which is not found in biological systems. Another solution to this conflict is just to make the different states of a molecule all equivalent in **Hyper-GO**. This would work, as there are no disjoint statements between the ChEBI classes, because chemical classification is compositional [51]. However, it makes some of the semantics wrong – acid would become a conjugate-base of itself, although it would now match the GO chemical classification. Similarly, GO also does not differentiate between a molecular entity and its naturally occurring forms, for instance, the *cobalt*

[2]https://www.ebi.ac.uk/QuickGO/term/GO:0015229

ion and its three common oxidation states: *cobalt(1+)*, *cobalt(2+)*, *cobalt(3+)* are considered as synonyms in GO. In Table 5.3 we summaries some of the chemical entities that have been used inconsistently in GO classes.

It would appear that GO is largely using a simplified representation: in the context of GO, we do not need to distinguish between the L-, D- nor generalised form of a molecule; likewise, the distinction between the acid and conjugate base are less relevant when, biologically, these molecules are always in solution. Therefore, a valid solution is to rely on the `cross-Ontology Relations` relation when the GO class label and definition have different chemical entities.

Table 5.3: Table showing some of the chemical entities that are used inconsistently within GO classes.

No	Molecule	No of classes in GO	Molecule states in ChEBI	Example
1	L-ascorbic acid	14	L-ascorbic acid or L-ascorbate or D-ascorbic acid	L-ascorbic acid metabolic process(GO:0019852)
2	succinate	31	succinate or succinate(2-) or succinate(1-) or succinic acid	succinate transport trans-porter(GO:0071422)
3	pantothenate	9	pantothenate or pantothenic acid	pantothenate metabolic process
4	uronic acid	4	uronic acid or uronate	uronate dehydrogenase activity
5	hexuronic acid	4	hexuronic acid or hexuronate	hexuronate transporter

5.3.5 Build the hypernormalised hierarchies

The hypernormalisation methodology aims to ease the process of ontology building, controlling its maintainability and increasing its expressiveness. To achieve this, the structure of a hypernormalised ontology should consists of trees of classes, divided into two main disjoint hierarchies. A set of self-standing classes represents the core classes of the ontology, and a set of refining or partitioning classes represents the biological qualities of those self-standing classes. The classes presented in a self-standing hierarchy are disjoint, and their sub-classes do not cover the superclasses,

whereas the hierarchy of the biological properties is expected to be comprehensive without any overlap between the classes (the methodology is explained in detail in Section 2.5.3). To enable a better understanding of the technique, it would be useful to show an example of how we can disentangle part of the GO hierarchy into two disjoint taxonomies. Figure 5.3 shows the current representation of the grouping class `response to stimulus (GO:0050896)`, which includes all the biological processes that describe a change in state or activity of a cell or an organism as a consequence of different types of stimuli (See Definition(6)).

Definition: 6. *Any process that results in a change in state or activity of a cell or an organism (in terms of movement, secretion, enzyme production, gene expression, etc.) as a result of an external stimulus.*
response to external stimulus (GO:0009605)

First we investigate the grouping class hierarchy, separating the biological process and non-biological process concepts and inspecting these concepts based on their ontological nature. Then, we build the hierarchy of the non-process classes. In this case, the set of *stimuluses* can be hierarchically represented as *refining classes* according to their classification in GO, while *self-standing classes* refers to the biological processes that describe an entire process of detecting a stimulus, responding to a stimulus type taking place in a cell or organism and the consequence of this stimulation. These processes will be defined in terms of the different stimuluses types in the refining hierarchy, as shown in Figure 5.4. Finally, the polyhierarchical relationships between the *self-standing classes* will be decided using a reasoner.

5.3.6 Pattern-driven development

Ontology Design Patterns (ODPs) play a vital role in facilitating development and enabling the accurate construction of ontologies by solving recurring design problems and producing a new style of ontology (discussed previously in section 2.6.1). Taking into account the class of `response to stimulus (GO:0050896)` discussed in the previous section. All the classes representing the processes of responding to a stimulus uses a standard definition (see Listing 5.6), only differing in the stimulus type. Therefore, we can speed up the development of the classes by designing a single pattern that can generate the relationships between the two independent hi-

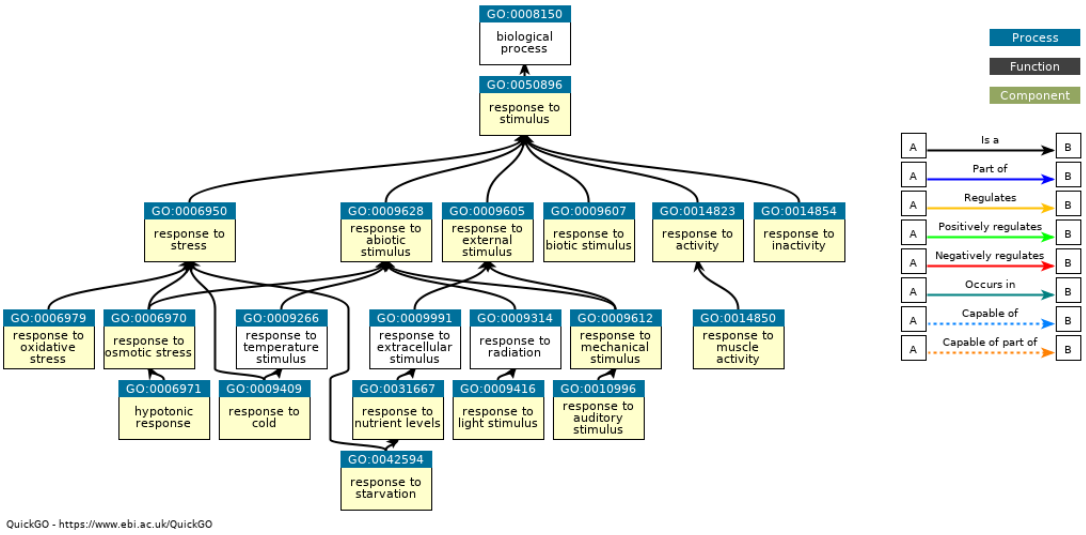


Figure 5.3: The process of responding to a stimulus, from QuickGO [15].

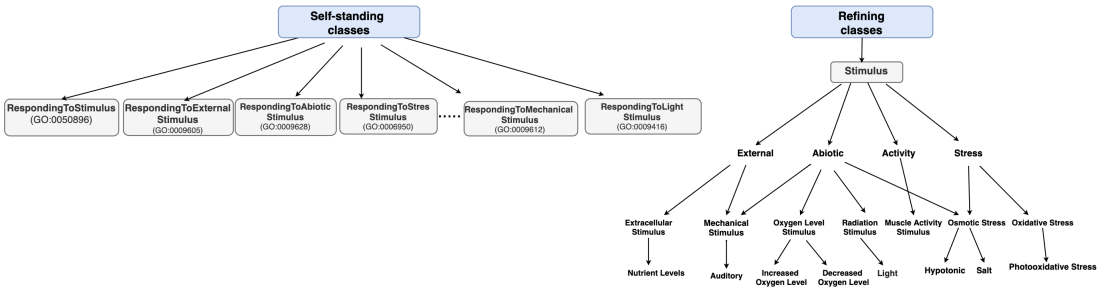


Figure 5.4: The hypernormalised hierarchy of the process responding to a stimulus.

erarchies automatically. A possible pattern to define a `response to stimulus` class is shown in Listing 5.7.

```
A change in state or activity of a cell or an organism
 (in terms of movement, secretion, enzyme production, gene
 expression, etc.) as a result of a stimulus type stimulus.
```

Listing 5.6: Response to stimulus (GO:0050896).

```
(defclass ResponsingToStress
  :annotation (goid "GO:0006950")
  :stimulus Stress
```

Listing 5.7: Response to stress.

5.3.7 Programmatic development

The idea of this project is to rebuild the ontology programmatically, so we can pull the ontology when we need it in a relatively straightforward and reproducible way. Many ontologies were developed using the Protégé editor, which has a rich visualisa-

tion interface but it is time and effort consuming. Tawny-OWL [76] (see Chapter 3) provides a set of patterns that explicitly support creation of a hypernormalised ontology.

5.4 Summary

In this chapter, we investigated the knowledge represented in the Molecular Function Ontology (MFO) as a preliminary step towards "untangling" the ontology hierarchy and then using the hypernormalisation approach to rebuild the ontology. Currently, MFO is formed from several large and tangled hierarchies of classes, each of which describe a broad molecular activity. The way to develop a hypernormalised hierarchy is to implement the methodology on each of the large grouping classes that form the MFO independently. We also discussed the set of steps we would take: starting with developing our biological knowledge about each activity and studying the logical representation of the activity classes, then analysing the textual definition associated with each class, because a class definition often describes the molecular function type, the biochemical entity that involved in the molecular function, and their biological properties.

During the investigation of the MFO, questions have been raised regarding the representations of classes in GO versus ChEBI. The main question is whether GO and ChEBI have different representations of classes for the same concepts. Theoretically, they describe the same entities, but actually GO often has a class, which represents a concept that does not directly map to a concept in ChEBI, normally because ChEBI is more specific and precise. The way to look at this is that as a meta-class, GO often refers to a more abstract version of the concept (the most common form), such as the *dehydroascorbic acid*, and considers its different naturally occurring forms (i.e., *L-dehydroascorbic acid*, *dehydroascorbide(1−) L-dehydroascorbate*) as equivalent classes. The class `dehydroascorbic acid transmembrane transporter, GO:0033300` in GO has a role *vitamin*; in ChEBI the role of *vitamin* is associated with the `L-dehydroascorbic acid, CHEBI:27956`, not the `dehydroascorbic acid, CHEBI:17242`. We have discussed the different solutions to tackle this issue, and their advantages and disadvantages. However, we found that the best

choice is to utilised the cross-ontology relationships that made by the GO-PLUS.

Finally, we have shown an example of how we can hypernormalise part of GO, specifically the grouping class `response to stimulus`. Moreover, we presented one possible logical pattern that could speed up the development of the stimulus classes using Tawny-OWL environment.

In the next chapter, we will apply the hypernormalisation and patternisation techniques on the Transporter Activity (TA).

6

Hypernormalisation of Transporter Activity (TA)

Contents

6.1 Introduction

`Transporter activity (GO:0005215)` is one of the broadest and complex grouping classes as it covers many cellular transporter systems by large number of ontology classes, constituting approximately 10% of all classes in MFO. The TA classes are grouped on the basis of functional similarity, which in general describe the molecular functions related to moving substances such as macromolecules, peptides, small molecules, and ions between cells, within a cell, into and out of a cell. There are several classified vital transport systems and mechanisms that are classified, by which substances or cellular entities travel across the membrane of a cell. Cell membranes are selectively permeable: each membrane of a cell has the ability to selectively allow essential molecules to pass through the membrane freely, while preventing certain molecules from entering a cell without the assistance of a membrane protein, cellular energy or both, for the purpose of regulating the cell's internal environment. For example, very small and non-polar molecules (e.g., water, oxygen and carbon dioxide) can move through a membrane's lipid bilayer unaided, whereas, water-soluble molecules (e.g., glucose, sodium (Na^+) and chloride (Cl^-)) are transported by means of membrane proteins, such as *channel* and *carrier proteins*. In addition, we can classify transport systems into those that are *passive transporters* and those that are *active transporters*. In GO, transporter systems are classified still further, which results in a tangled ontological hierarchy; as such, any required development or maintenance to the hierarchy will require the maintenance of multiple classifications. Therefore, the development of a hypernormalised hierarchy of TA is potentially useful for easing development and maintenance, and providing a strong computational solution to recurrent modelling problems.

For our approach to hypernormalise the `TA (GO:0005215)` hierarchy, we follow the steps illustrated in Chapter 5. We start by acquiring sufficient knowledge covering the different membrane transport systems, the elements that influence the transfer, the different cell types and the parts that maintain a specific set of transport proteins and the physicochemical properties of the substances (e.g., hydrophilic, charged, polar and large molecules) that being transported. Taking the structural representation and the scope of the TA classes into account, we consider what this

does and does not cover. Then, we analyse the textual description associated with each TA class, as definitions include further explanations for the type of transport, the initial location of a substance and the final position (this represents a simple transport system, whereas, in more complicated transport systems, other biological and chemical elements are essential to perform the function). Finally, we restructure the hierarchy of the transporter activity into disjoint trees, *self-standing classes* and *refining classes* with the assistance of a set of designed patterns and logical reasoners, to build the TA sub-hierarchies.

In the section that follows, we will introduce the range of membrane transport systems and mechanisms represented in the TA classes, the set of essential elements and characteristics associated with each type and the way in which they organised the TA hierarchy.

6.2 Reviewing TA structure and classes

The total number of classes defined within the TA hierarchy is around 1,065 classes, however, the number of classes change based on the changes made by the GOC ontology team and scientists. Our analysis shows that almost all of the TA classes are classified under the class `transmembrane transporter activity (GO:0022857)` [1] which is a grouping class (i.e., not a function itself) with a general definition (see Definition(7)), though its subclasses are more specific. The subclasses are distributed over three categories: general classifications for narrower transport classes, *passive transport* and *active transport*.

Definition: 7. *Enables the transfer of a substance, usually a specific substance or a group of related substances, from one side of a membrane to the other.*
transmembrane transporter activity (GO:0022857)

It is generally held among cellular biologists that there are two major ways in which substances can move across a cell membrane: *passive transport* and *active transport* [1]. Passive transport can be further classified into three transport methods: *simple diffusion*, *osmosis* and *facilitated diffusion*. Active transport is also sub-divided into two types: *primary active transport* and *secondary active trans-*

[1] We use different font styles for a concept, a class and label, see Section 2.8

port. In GO, the classes representing active transport functions are grouped using the `active transmembrane transporter activity (GO:0022804)` which has the sub-groups classes: `primary active transmembrane transporter activity (GO:0015399)`, and `secondary active transmembrane transporter activity (GO:0015291)`. Whereas, the set of classes referring to the transport functions that occur by means of the *facilitated diffusion* transport system are defined under the grouping class of `channel activity (GO:0015267)`, which in turn, is classified as a *passive transport*. However, in regards to the *simple diffusion* transport system, this is not within the scope of GO as it is not enabled by any membrane protein; it just describes substances passing through the cell membrane. TA's overall structure and the main grouping classes that define the set of transport mechanisms are illustrated in the list below 6.2. These are not the only grouping classes; there are narrower and chemical type-based groups. For example, the grouping class `macromolecule transmembrane transporter activity (GO:0022884)`, is defined to represent any large molecule, such as a protein or nucleic acid, but in **Hyper-GO**, this type of grouping will be classified based on the ChEBI classification.

i) **[GF] transporter activity (GO:0005215)**

   a) **[GF] transmembrane transporter activity (GO:0022857)**

      1) [SF] lactone transporter activity (GO:0042971)
      2) [SF] amide transporter activity (GO:0042887)
      3) [SF] toxin transmembrane transporter activity (GO:0019534)
      4) **[GF] active transporter activity (GO:0022804)**

         (i) **[GF] primary active transporter activity (GO:0015399)**

            (a) [GF] ATPase-coupled transmembrane transporter activity (GO:0042626)

            (b) [SF] light-driven active transmembrane transporter activity (GO:0015454)

            (c) [SF] decarboxylation-driven active transmembrane transporter activity (GO:0015451)

(d) [SF] P-P-bond-hydrolysis-driven protein transmembrane transporter activity (GO:0015450)

(ii) **[GF] secondary active transporter activity (GO:0015291)**

(a) [GF] symporter activity (GO:0015293)

(b) [GF] uniporter activity (GO:0015292)

(c) [GF] antiporter activity (GO:0015297)

(d) [SF] proton-dependent peptide secondary active transmembrane transporter activity (GO:0022897)

5) **[GF] passive transporter activity (GO:0022804)**

(i) [GF] channel activity (GO:0015267)

b) [GF] lipid transporter activity (GO:0005319)

c) [SF] protein transporter activity (GO:0140318)

Top-level Structure 6.1: This is the top-level classes in the hierarchy of TA which represent the major cellular transport systems. The GF refers to group function, whereas SF refers to single function

In the following subsections, we will describe the major types of cellular transport systems and the set of biological and electrochemical elements that influence each category. It is an essential step if we are to divide the TA classes into two main hierarchies: *self-standing classes* and *refining classes*.

6.3 Active transport

A large number of classes describe the active transport of solutes, approximately 418 of the total TA classes [2]. *Active transport* is the movement of substances from an area of low concentration to an area of higher concentration (i.e., against an electrochemical gradient); that can be inside a cell, within a specific cellular component or outside of a cell environment. The electrochemical gradient of a substance refers to the differences in both the electrical and chemical concentration gradients of that substance, for outside a cell versus inside. For the substances that move against its concentration gradient and across the cell membrane, a cellular energy is required,

[2]We use the web-based open-source tool GOOSE to query and retrieve GO data see Section 2.4

often coming from the assistance of special membrane carrier proteins. The energy is usually either released by the hydrolysis of Adenosine Triphosphate (ATP) or from stored in a substance gradient that was created as a result of using the ATP energy. Therefore, the active transport classes are classified into two main types: *primary* and *secondary active transport*.

6.3.1 Primary active transport

Primary active transport refers to the transport of molecules (such as sodium(Na^+), potassium(K^+), magnesium($Mg2^+$) and calcium($Ca2^+$)) that are assisted by carrier proteins (known as pumps) to cross the cell membrane against their concentration gradients, with the help of free energy released from ATP hydrolysis. One of the best-known example is the sodium-potassium pump, where cells utilise energy from ATP to pump sodium ions out of the cell while bringing a potassium ion into the cell. In both cases, the ions are pumped against a concentration gradient, with Na^+ having a high concentration outside and K^+ having a high concentration inside the cell (see Figure 6.1). The sodium ions transported out of the cell form a potential energy that can later be exploited as an energy source for secondary active transport. This does not mean that ATP hydrolysis is the only primary active transport; there are other prime energy sources, such as the following chemical reactions: *oxidoreduction*, *methyl transfer*, *decarboxylation* and *light absorption*. However, these types of energy are used only in certain organisms, whereas ATP-dependent transporters are identified in all living organisms and a large portion of the primary active transporter classes in TA are ATP-dependent.

6.3.2 Secondary active transport

The sodium-potassium pump is important not only for stabilising the resting membrane potential of a cell and its volume, but also because of many secondary active transporters. Secondary active transport, also known as ion-coupled transport, refers to the utilisation of the energy held in the electrochemical gradient of a molecule or an ion (created by primary active transport) to transfer other molecules against their concentration gradient and across a cell membrane. In other words, specific carrier proteins allow certain molecules, such as glucose or an amino acid, to move across

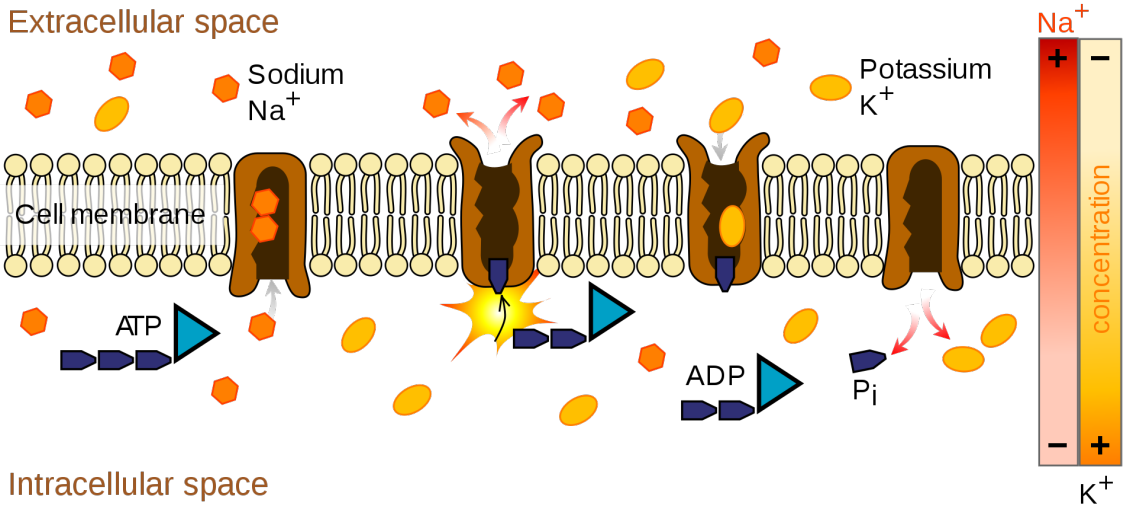


Figure 6.1: An example of primary active transport, using ATP as an energy source to implement the sodium-potassium pump cycle. Figure taken from https://en.wikipedia.org/wiki/Active_transport#Primary_active_transport , used under CC-BY 4.0 license.

the cell membrane and up their concentration gradient only by coupling them with another ion (typically sodium (Na^+) or a proton (H^+)), which moves down its electrochemical gradient. An example of a secondary active transport is when a sodium ion drives the transport of an amino acid out of a cell by binding to a carrier-specific protein in a tightly coupled process (see Figure 6.2). The direction of translocated substances depends on the type of the active membrane carrier protein; these are classified as *symporters*, *antiporters* or *uniporters*.

- Symporter (also known as cotransporter): The driving and the driven substances both move in the same direction.
- Antiporter (also known as counter-transport or exchanger): Moves substances in opposite directions.
- Uniporters: Transports a single molecule or ion from one side of a cell membrane to the other.

One limitation of the secondary active transporters' representation in GO is that the secondary energy-coupling source is not explicitly stated. In many cases, the ontology definitions of secondary active transporters indicate that the movement of secondary transporters is driven by "chemiosmotic sources of energy". These are ion gradients (i.e., an ion electrochemical gradient) that can be found on both sides of

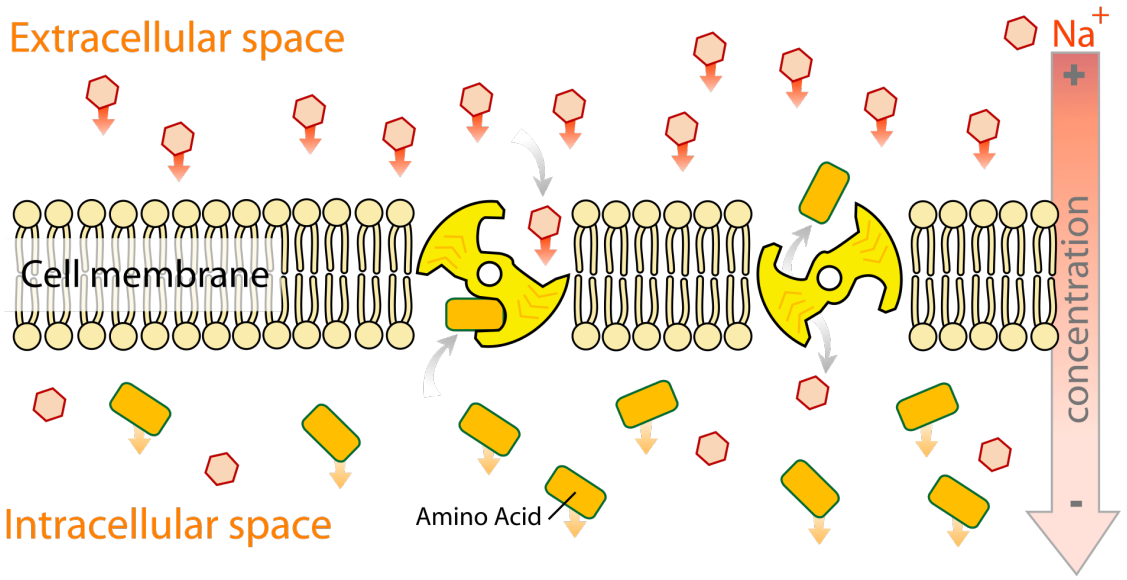


Figure 6.2: An example of secondary active transport: the transport of amino acid driven by sodium ion. Figure taken from https://en.wikipedia.org/wiki/Active_transport#Secondary_active_transport , used under CC-BY 4.0 license.

a cell membrane, such as Na^+, K^+, H^+ and $Ca2^+$ gradients, which are utilised by membrane proteins to move ions or molecules out of or into a cell. In many different cells and tissues, a sodium ion functions as the driving ion for many secondary active transporters, while some other co-transporters use an existing proton electrochemical gradient to drive the uphill transport of other solutes. An additional limitation is that the representation of symporter and antiporter transport mechanisms can lead to confusion, as it does not make a clear distinction between the solute being transported and the driving ion (i.e., which solute is driving and which is driven). For instance, consider the symporter class `anion:cation symporter activity (GO:0015296)` in Definition(8) and the antiporter class `calcium:sodium antiporter activity (GO:0005432)` in Definition(9).

Definition: 8. *Enables the transfer of a solute or solutes from one side of a membrane to the other according to the reaction: anion(out) + cation(out) = anion(in) + cation(in).*
anion:cation symporter activity (GO:0015296)

Definition: 9. *Enables the transfer of a solute or solutes from one side of a membrane to the other according to the reaction: Ca2+(in) + Na+(out) = Ca2+(out) + Na+(in). PMID:16371597*
calcium:sodium antiporter activity (GO:0005432)

Another issue is consideration of the uniporter transporters as secondary active

transporters, which is inaccurate. Uniporters are integral membrane proteins that facilitate the diffusion of charged molecules using the electrical potential difference across a cell membrane; they do not require direct cellular energy or a driving substance to function, according to a paper by Milton Saier [98]. The paper is included as a source of information in the definition of the class of `secondary active transport (GO:0015291)` and considered to be a common reference for the transmembrane molecular transport systems. Also, the [122] and [74] state that uniporter membrane proteins are facilitated transporters, not active transporters. More importantly, the definition of the class of `uniporter activity (GO:0015292)`, holds the term *facilitated diffusion carrier* as an exact synonym.

6.4 Passive transport

Another substantial number of the TA classes describe the passive transport of molecules through transmembrane channel and carrier proteins by *facilitated diffusion*. In passive transport, solutes move across a cell membrane down their concentration gradients with no expending of any cellular energy. Passive transporters are divided into simple and facilitated diffusions:

6.4.1 Simple diffusion

It is the simplest and most direct transport system, by which hydrophobic, uncharged, small and non-polar molecules are able to dissolve in the phospholipid bilayer and pass through the cell membrane. The passage of these molecules will continue into and out of the cell's semi-permeable membrane until the concentration gradient distribution of the solutes becomes even. A special case of simple diffusion is *osmosis*, which is the free transport of water across the cell membrane and along its concentration gradient. However, this type of transport is not in the scope of GO as it does not involve transporter proteins.

6.4.2 Facilitated diffusion

For those substances that are not able to pass through the cell membrane as they are repelled by the hydrophobic part of the phospholipid bilayer, membrane transport proteins are required to facilitate their diffusion [3]. As such, facilitated transport

enables hydrophilic, charged, polar and large molecules, such as glucose, amino acids and ions, to move across the membrane using channel proteins and carrier proteins.

Channel Proteins

Channels are membrane-embedded proteins that form hydrophilic tunnels through which polar, charged and a specific size of molecules can avoid the hydrophobic tails and move from one side of a membrane to the other [29]. Channel proteins are dedicated to a specific solute or one group of similar solutes, which they can transport through freely and rapidly. For instance, water channel proteins (known as *aquaporins*) specifically allow the movement of water. Many types of ion are transported across the cell membrane via different types of *ion channel*. However, not all channels are open all of the time; there are channels known as *gated channels* that only open in response to a stimulus. Gated channels allow cells, such as nerve and muscle cells, to control the flow of electrical signals by regulating the opening and closing of their ion channels. There are several stimuli that affect the opening and closing of gated ion channels:

- *Voltage-gated channels*: The set of channels that open and allow the transmembrane transfer of a substance in response to changes in the electric potential across the cell membrane. These channels are often ion-specific, such as voltage-gated chloride channels and voltage-gated sodium channels.
- *Ligand-gated channel*: A class of ion channels that enable the transmembrane transfer of solutes only when chemical signals (i.e., ligands) binds to the channels' receptors to unlock their pores. There can be two types of ligands: intracellular ligands (that bind to receptors inside the cell, e.g., glutamate) or extracellular ligands (that bind to receptors outside the cell, e.g., calcium ions ($Ca2^{+}$)).
- *Mechanically-gated channels*: A group of channels that open in response to a physical pressure or stress.

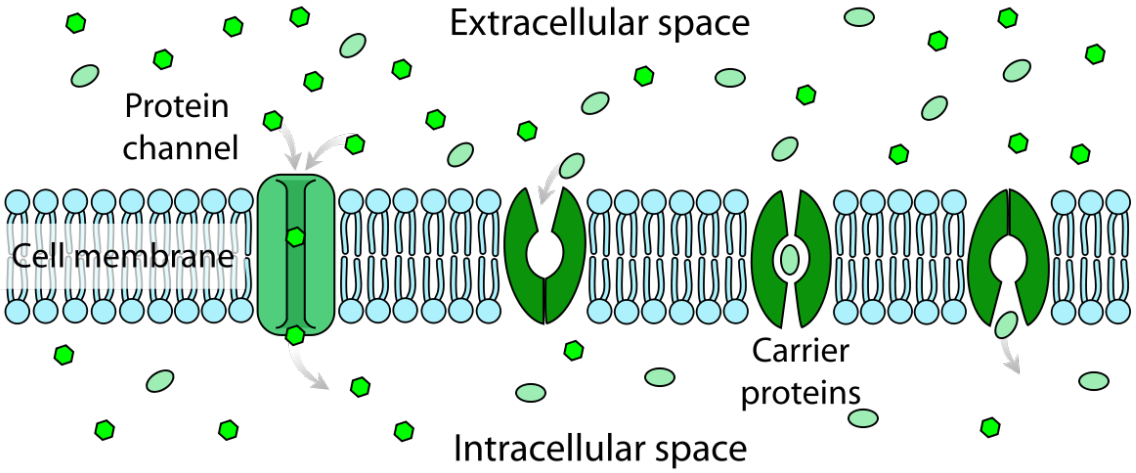


Figure 6.3: An example of a facilitated diffusion: the transport of materials using channel proteins versus carrier proteins. Figure taken from https://en.wikipedia.org/wiki/Facilitated_diffusion, used under CC-BY 4.0 license.

Moreover, special channel proteins allow the passage of substances between adjacent cells through *gap junctions*, which can take the form of wide or narrow pore channels.

Carrier Proteins

Carrier proteins are another type of cellular membrane-embedded protein that require a binding process with other substances, to facilitate their selective diffusion through a cell membrane. Each carrier protein goes through changes in its conformation each time polar and charged molecules bind to the carrier binding site, which is specific. In fact, both passive and active transport systems use structurally similar carrier proteins to transfer solutes across the membrane, but with differences in the direction of the solutes' concentration gradients. The difference between channel and carrier proteins is that substances moves much more slowly using carrier proteins compared to channel proteins. This is because channels do not change their shape in order to transfer molecules. In addition, channels only transfer solutes passively via facilitated diffusion, unlike carrier proteins, which transport solutes actively and passively. Figure 6.3 shows the difference between channel and carrier proteins in transporting materials.

Although carrier proteins are broadly recognised as membrane transporter

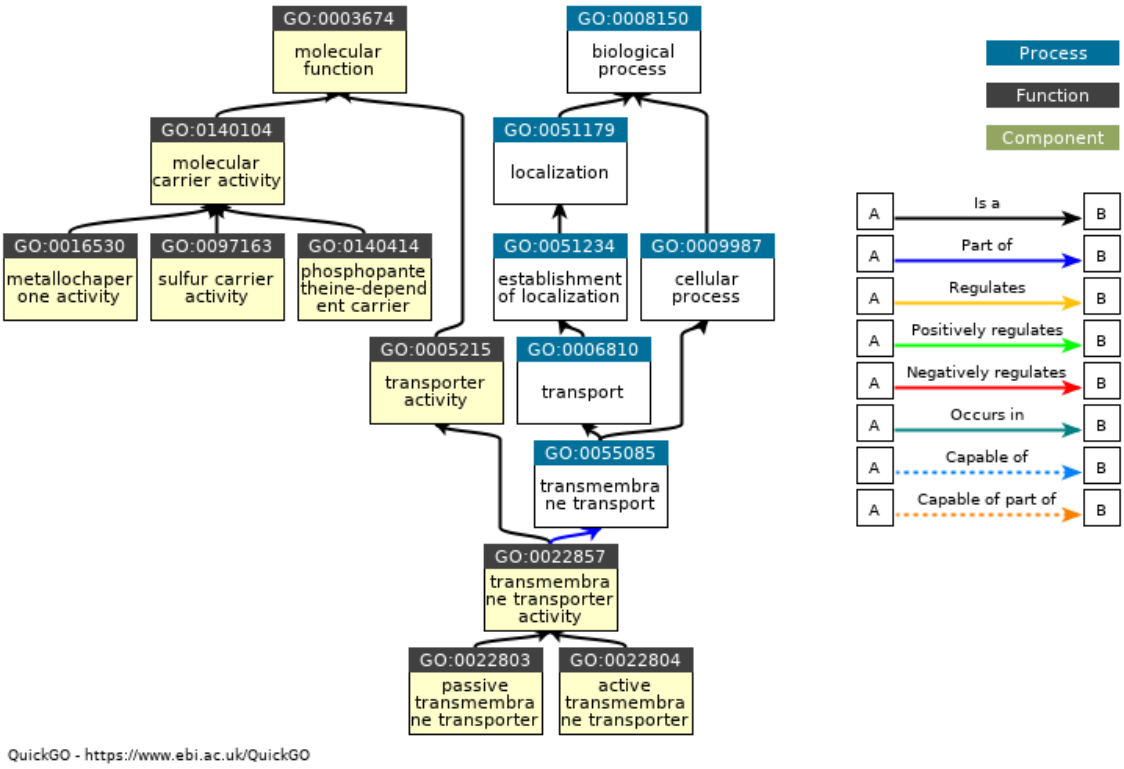


Figure 6.4: The GO hierarchical representation of transporter activity and carrier activity

proteins [2] and assist in performing active and passive transporter systems, the GO developers draw a distinction between a "transporter" and a "carrier". A transporter forms a pore that allows molecules to move through, whereas a carrier binds to substance and delivers it to a cellular location. As a result of this distinction, the set of solutes that is transported using a carrier protein is defined outside of the `transporter activity (GO:0005215)` hierarchy, in an independent hierarchy of `molecular carrier activity (GO:0140104)` (see Figure 6.4). Despite that, we find a number of carrier classes defined with the TA hierarchy, such as `ceramide transfer activity (GO:0120017)`.

6.5 Summary of the TA review

In the previous sections we covered the major classes of membrane transport proteins, the mechanism by which each protein works, the physicochemical properties of the substances that are allowed to be transported and the biological roles of stimuli and energy sources. This allows us to understand the biological concepts defined within the grouping class of `transporter activity (GO:0005215)`. It is

an essential step as the GO representation is developed by expert biologists with a brief description of the cellular transporter activities. In Table 6.1, we summarise the transmembrane molecular transport systems defined within the TA hierarchy, as well as the proportion of each system. In addition, we have shown the differences in the representations between GO and external resources for some of the transporter systems such as secondary active transporters, uniporters and carriers.

Groups	No.classes	Percentage
General classification	417	39.15%
Energy-dependent transport: active transport	414	38.87%
Energy-independent transport: facilitated diffusion	218	20.46%
Others (Obsolete and duplicate terms)	16	1.5%
Total	1065	100%

Table 6.1: The three main transport categories that comprising the hierarchy of the transporter activity

In the sections that follows, we will present our development of a hypernormalised transporter activity in our ontology, **Hyper-GO**. The implementation is available at `https://github.com/phillord/hyper-go`. Before we do that, we need to clarify the difference between the biological process of `transport (GO:0006810)`, and the molecular function of `transporter activity (GO:0006810)` (see Figure 6.5). In general, a biological process in GO refers to any biological objective (e.g., cell growth or bone mineralization) that can be achieved by multiple molecular functions. However, the definitions of the transport classes as biological processes include no different description than their corresponding molecular function classes in the transporter activity. For instance, consider the biological process of `nucleoside transmembrane transport (GO:1901642)`, in Definition(10) and the molecular function of `nucleoside transmembrane transporter activity (GO:0005337)`, in Definition(11)

Definition: 10. *The directed movement of nucleoside across a membrane.*
nucleoside transmembrane transport (GO:1901642)

Definition: 11. *Enables the transfer of a nucleoside, a nucleobase linked to either beta-D-ribofuranose (ribonucleoside) or 2-deoxy-beta-D-ribofuranose, (a deoxyribonucleotide) from one side of a membrane to the other.*
nucleoside transmembrane transporter activity (GO:0005337)

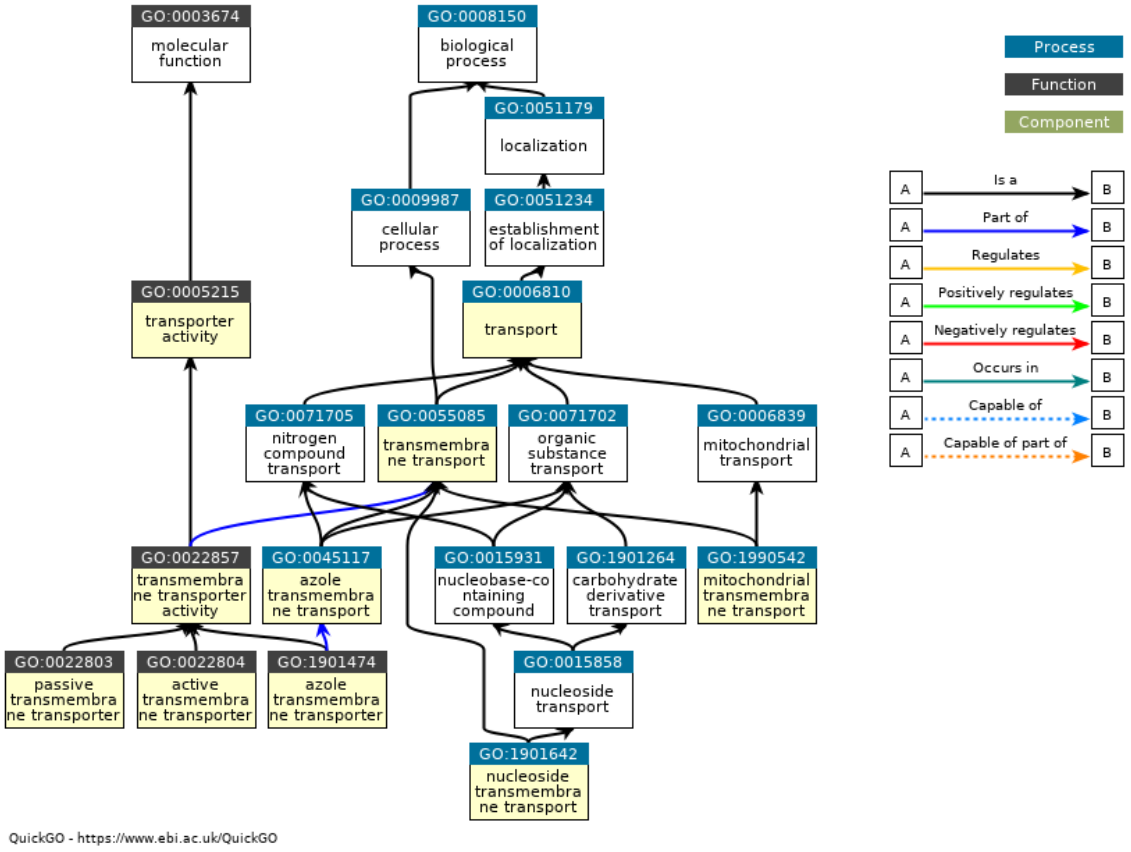


Figure 6.5: The GO representation of transporter activity MF and transport BP.

That is, the hypernormalisation of the biological transport process in current representation would not be beneficial as the difference between the biological transport process and the molecular function transporter activity is not obvious. Nevertheless, to make explicit separation between a molecular function and a biological process, we used the infinitive of transport with a prefix 'to' to represent the function classes (i.e., *ToTransport*) and a gerund to represent any future development of a biological process classes (i.e., *Transporting*).

6.6 Hyper-TA development

The main tool we have used to develop our ontology is the Tawny-OWL [76] library, which enables the construction of OWL ontologies and provides a set of patterns that explicitly support the creation of a hypernormalised ontology (see Chapter 3). We have extended the Tawny-OWL frame-based syntax by defining new entities such as `deftransport` and `defcatalyse` to ease the creation of our ontology classes. The `deftransport` entity is a new class with some property restrictions such as

`transports`, `transports-from` and `driven-by`. That is to say, we extend the basic Tawny-OWL frames with new frames that are restricted to the entities we developed (i.e., `deftransport` or `defcatalyse`). In Table 6.2, we show the created frames and their defined equivalent object properties. An example of a TA class created using the `deftransport` pattern is shown in Listing 6.1.

```
(deftransport ToTransportPhospholipid
  :annotation (goid "GO:0005548")
  :cargo ch/phospholipid)
```

Listing 6.1: The definition of the `phospholipid transporter activity (GO:0005548)`.

Object property	Frame
transports	:cargo
transports-across	:across
transports-from	:from
transports-to	:to
driven-by	:driven
hasDirection	:direction
hasTransporterAffinity	:transporterAffinity
linked-to	:linked
dependent-on	:when
hasTransportSystem	:system
hasIonTransportMechanism	:ion-mechanism
involved_in	:involved
occurs_in	:occurs
hasMembraneTransportProtein	:via
contributeTo	:contribute
hasReactant	:reactant
hasProduct	:product
hasEnzymeClass	:enzyme

Table 6.2: List of the created properties and their equivalent frames.

6.6.1 General classification

The first category of transport classes in Table 6.1 does not follow a common cellular transport system. They are of a general classification, which describes the transport of substances (typically chemical entities) from one side of a membrane to the other, with no energy source or facilitator proteins required. Nearly half of the transporter activity classes are of a general classification for narrower transport classes. As a

result, we designed a *content-specific pattern*, see Section 2.6.1), that can generate these classes using Clojure functions. In Listing 6.2 we show the general transport pattern that is designed to generate these classes, which need to be presented in an organised list, is shown in Listing 6.3.

The minimum data for any transporter activity, regardless of the transport type, are:

1. Substance: The GO name of the substance transported (e.g., macromolecules, peptides, small molecules, or ions)

2. GO ID: The Gene Ontology ID for each class (e.g., GO:0000001)

3. Equivalent ChEBI class: The corresponding substance from the ChEBI ontology.

The designed pattern reads transporter data from Listing 6.3, starting with the transported solute's name and GO ID from the class description in GO, the cellular component `membrane (GO:0016020)` that a solute `transport-across` (from the CCO) and the equivalent ChEBI class, which can have zero, one or multiple properties. The reason for having the conditional statement (here *cond*) is to maintain the correct definitions of classes, properties and their values, as solutes can have different properties (i.e., biological, chemical and application) and for some solutes, a property may have multiple values. For example, if we take the class, `L-proline transmembrane transporter (GO:0015193)`, it `has an application role` as a `drug`, `has acidity` as a `Natural` and has `enantiomerism` as a `L-enantiomer`. The `L-proline` class is generated from the developed pattern and the result is shown in List 6.4. In the case that a class description is very complex, it will be defined individually.

```
(defn substance-transporting-transmembrane [lis]
`(deftransport ~(symbol (str "ToTransport" (first lis) "
   Transmembrane"))
  :annotation (goid ~(second lis))
  :across go/membrane
  :cargo
  ~(cond
    (= 3 (count lis)) (nth lis 2)
    (= 5 (count lis)) `(owl-and ~(nth lis 2)
                        (owl-some ~(nth lis 3) ~(nth lis 4)))
    (= 6 (count lis)) `(owl-and ~(nth lis 2)
           (owl-some ~(nth lis 3) ~(nth lis 4) ~(nth lis 5)))
    (= 7 (count lis)) `(owl-and ~(nth lis 2)
                       (owl-some ~(nth lis 3) ~(nth lis 4))
                       (owl-some ~(nth lis 5) ~(nth lis 6)))
    (= 8 (count lis)) `(owl-and ~(nth lis 2)
           (owl-some ~(nth lis 3) ~(nth lis 4))
           (owl-some ~(nth lis 5) ~(nth lis 6) ~(nth lis 7)))
    (= 9 (count lis)) `(owl-and ~(nth lis 2)
                      (owl-some ~(nth lis 3) ~(nth lis 4))
                      (owl-some ~(nth lis 5) ~(nth lis 6))
                      (owl-some ~(nth lis 7) ~(nth lis 8)))
    (= 10 (count lis)) `(owl-and ~(nth lis 2)
                      (owl-some ~(nth lis 3) ~(nth lis 4))
                      (owl-some ~(nth lis 5) ~(nth lis 6))
                      (owl-some ~(nth lis 7) ~(nth lis 8)
                      ~(nth lis 9)))
    (= 11 (count lis))`(owl-and ~(nth lis 2)
                       (owl-some ~(nth lis 3) ~(nth lis 4))
                       (owl-some ~(nth lis 5) ~(nth lis 6))
                       (owl-some ~(nth lis 7) ~(nth lis 8))
                       (owl-some ~(nth lis 9)~(nth lis 10)))
    (= 12 (count lis))`(owl-and ~(nth lis 2)
                       (owl-some ~(nth lis 3) ~(nth lis 4))
                       (owl-some ~(nth lis 5) ~(nth lis 6))
                       (owl-some ~(nth lis 7) ~(nth lis 8))
                       (owl-some ~(nth lis 9)~(nth lis 10)
                       ~(nth lis 11)))
   :else (println "Class data out of the range"))))

;; macro function to do the classes mapping
(defmacro deftransporters-transmembrane [& lis]
  `(do ~@(map substance-transporting-transmembrane lis)))
```

Listing 6.2: The general transmembrane transport of solutes pattern.

```
(deftransporters-transmembrane
  ["Substance" "GO:0022857" ch/chemical_entity]
  ["Drug" "GO:0015238" ch/chemical_entity
     has-application-role ch/drug]
  ["Vitamin" "GO:0090482"         ch/chemical_entity
     has-biological-role ch/vitamin]
  ["CarbonDioxide" "GO:0035379" ch/carbon_dioxide
     has-application-role  ch/drug]
  ["L-histidine" "GO:0005290"   ch/L-histidine  hasAcidity
           Alkaline         has-application-role    ch/drug]
  ["Azole" "GO:1901474" ch/azole]
  ["Fluconazole"          "GO:0015244"  ch/fluconazole
     has-biological-role ch/antimicrobial_drug   ch/xenobiotic
     ]
  ["Acadesine" "GO:1903089"      ch/acadesine
     has-application-role         ch/drug]
  ["ThiaminePyrophosphate" "GO:0090422"  ch/thiamine_1+
     __diphosphate_1-_      has-biological-role         ch/
     vitamin]
  ......
  )
```

Listing 6.3: A TA class is defined in a list with all the properties that describe a molecular transporter function.

```
(transports some
    (L-proline
     and (has-application-role some drug)
     and (hasAcidity some Neutral)
     and (hasEnantiomerism some L-Enantiomer)))
 and (transports-across some membrane)
```

Listing 6.4: The class `L-proline transmembrane transporter (GO:0015193)` generated from the pattern in Listing 6.2.

There are a number of transport classes that specify the starting cellular location of the transport process and the final location. These classes are defined in a pattern that does not differ greatly from the previous pattern, except it includes the `transports-from` and `transports-to` object properties.

6.6.2 Energy-independent transport: Facilitated diffusion

Both simple and facilitated diffusion are passive transporters, however, the latter requires the assistance of membrane-bound proteins to allow solutes with certain physicochemical properties to pass through the cell membrane. Channel and carrier proteins are the facilitator proteins, although the latter has not been classified as a

means of passive transport in the TA hierarchy. The definitions of the facilitated diffusion classes are not as simple as the first category as there are distinct elements influencing the transport process. The simplest definition of a substance transported through a channel, can be defined in the same way that `methylammonium channel activity (GO:0015264)`, is defined in Listing 6.5.

```
(deftransport ToTransportAmmoniumIonByChannel
  :annotation (goid "GO:0015264")
  :across go/membrane
  :system FacilitatedDiffusion
  :via Channel
  :cargo (owl-and ch/methylammonium
           (owl-some hasConcentration HighConcentration)))
```

Listing 6.5: The facilitated diffusion of methylammonium by channel.

For the set of classes that describes the movement of solutes through a gated channel that opens in response to a specific stimulus, including the `hasStimulus` property and the stimulus, see Listing 6.6. The set of stimuli defined using the `defpartition` pattern (see Section 3.3) is considered as a refining type. Broadly speaking, the transport classes are created with properties that are related to the transport system (i.e., facilitated channels or active transport (primary or secondary)), in addition to the properties(e.g., biological roles) of the transported solutes. When it comes to the GO ID and Identitas ID, the latter will be generated randomly for all classes defined in the **Hyper-GO** ontology.

```
(deftransport
   ToTransportSoluteByGatedChannelInResponseToStimulus
 :annotation (goid "GO:0022836")
 :across go/membrane
 :system FacilitatedDiffusion
 :via (owl-and Channel
        (owl-some hasStimulus
         (owl-or Stimulus ch/chemical_entity)))
 :cargo (owl-and ch/chemical_entity
              (owl-some hasConcentration HighConcentration)))
```

Listing 6.6: The facilitated diffusion of a chemical entity through a channel in response to a stimulus.

```
(defpartition Stimulus
  [Osmolarity MechanicalStress Voltage
  IntermediateVoltage Light volume-sensitive Phosphorylation
   Dephosphorylation inward-rectification temperature
   EmptyingOfIntracellularCalciumStores]
   :domain Channel
   :super ChannelStimulus
   :comment "Gated channel: enables the transmembrane
   transfer of solute by a channel that opens in response
   to a specific stimulus.")
```

Listing 6.7: The set of identified stimuli that open gated channels.

More specific classes of gated channels contribute to the regulation of a qualitative or quantitative trait of a biological quality. For example, consider the class of `stretch-activated, cation-selective, calcium channel activity involved in the regulation of action potential (GO:0097364)`, in Definition(12), and its corresponding implementation in Listing 6.8. In our ontology, we made use of the BPO and CCO classes by linking the channel classes that contribute to the modulation of biological processes and/or that occur in some cellular component.

Definition: 12. *Enables the transmembrane transfer of a calcium ion by a channel that opens in response to a mechanical stress in the form of stretching, and contributing to the regulation of action potential.*
stretch-activated, cation-selective, calcium channel activity involved in regulation of action potential (GO:0097364)

```
(deftransport
   ToTransportMechanicalSensitiveCalciumIonByChannel
   InvolvedInRegulationOfActionPotential
  :annotation (goid "GO:0097364")
  :comment "Involved in regulation of action potential"
  :across go/membrane
  :system FacilitatedDiffusion
  :involved go/regulation_of_action_potential
  :via (owl-and Channel (owl-some hasStimulus
     MechanicalStress))
  :cargo (owl-and ch/calcium_ion (owl-some hasConcentration
     HighConcentration)))
```

Listing 6.8: The facilitated diffusion of a calcium ion by a channel that opens in response to a mechanical stress and involved in regulation of action potential.

`Ligand-gated channel activity (GO:0022834)`, is a grouping class that describes the transmembrane transfer of solutes through channels that open/close in

response to the binding of a ligand (also known as a chemical messenger). Ligands are ions, molecules, or molecular groups that functions like keys for the gated channels, opening and closing them by binding to the receptors located on the plasma membrane. For instance, `histamine-gated chloride channel activity (GO:0019182)` describes the transport of a chloride ion through a channel after the ligand (i.e., here histamine) has attached to the receptor; this is defined in Listing 6.9.

```
(deftransport ToTransportChlorideByHistamineGatedChannel
 :annotation (goid "GO:0019182")
 :across go/membrane
 :system FacilitatedDiffusion
 :via (owl-and Channel (owl-some hasStimulus
          (owl-and ch/histamine
           (owl-some has-biological-role ch/neurotransmitter)
         (owl-some bound-by go/membrane_receptor_complex))))
 :cargo (owl-and ch/chloride (owl-some hasConcentration
    HighConcentration)))
```

Listing 6.9: The facilitated diffusion of chloride ion by a channel that opens when histamine has been bound by the channel complex or one of its constituent parts.

Glutamate receptor, *nicotinic acetylcholine receptor* and *glycine receptor* are some of the well-known ligand-gated ion channel receptors that allow the binding of neurotransmitters and the transmission of ions. For instance, consider the class `AMPA glutamate receptor activity (GO:0004971)` in Definition (13); this is defined in Listing 6.10 in our ontology.

Definition: 13. *An ionotropic glutamate receptor activity that exhibits fast gating by glutamate and acts by opening a cation channel permeable to sodium, potassium, and, in the absence of a GluR2 subunit, calcium.*
AMPA glutamate receptor activity (GO:0004971)

```
(deftransport ToTransportCationByGlutamateAMPAGatedChannel
 :annotation (goid "GO:0004971")
 :across go/membrane
 :system FacilitatedDiffusion
 :via (owl-and Channel (owl-some hasStimulus
        (owl-and ch/L-glutamic_acid
         (owl-some has-biological-role ch/neurotransmitter)
         (owl-some bound-by go/
            AMPA_glutamate_receptor_complex))))
 :cargo (owl-and ch/cation (owl-some hasConcentration
    HighConcentration)))
```

Listing 6.10: The facilitated diffusion of cation by a channel that opens when L-glutamic_acid has been bound by the AMPA glutamate receptor complex.

A group of substances move across specific types of channels, such as wide and narrow pores, such as the `potassium ion leak channel activity (GO:0022841)`, `wide pore channel activity (GO:0022829)` and `gap junction channel activity (GO:0005243)`; they are defined in Listing 6.11, Listing 6.12 and Listing 6.13 respectively. In addition, porins [99] are another membrane proteins in bacterial cells that allow the facilitated diffusion of small hydrophillic molecules with specific size (e.g., <1000 Da or <600 Da) to pass through the cell membranes. Listing 6.14 shows an example of a class definition in which the solute `maltose` has size less than 1000 Da from one side of a membrane to another.

```
(deftransport ToTransportPotassiumByNarrowPoreChannel
 :annotation (goid "GO:0022841")
 :across go/membrane
 :system FacilitatedDiffusion
 :via (owl-and Channel (owl-some hasChannelType
    NarrowPoreChannel))
 :cargo (owl-and ch/potassium_1+_ (owl-some hasConcentration
    HighConcentration)))
```

Listing 6.11: The transport of a potassium ion across a membrane via a narrow pore channel.

```
(deftransport ToTransportSoluteByWidePoreChannel
 :annotation (goid "GO:0022829")
 :across go/membrane
 :system FacilitatedDiffusion
 :via (owl-and Channel
         (owl-some hasChannelType WidePoreChannel))
 :cargo (owl-and ch/chemical_entity
              (owl-some hasConcentration HighConcentration)))
```

Listing 6.12: The transport of a solute across a membrane via large pore channel.

```
(deftransport
   ToTransportSoluteByGapJunctionWidePoreChannelChannel
 :annotation (goid "GO:0005243")
 :across go/membrane
 :system FacilitatedDiffusion
 :from go/cell
 :to go/cell
 :via (owl-and Channel
        (owl-some hasChannelType Gap-junction))
 :cargo (owl-and ch/chemical_entity
             (owl-some hasConcentration HighConcentration)))
```

Listing 6.13: The transport of solute through a gap junction.

```
(deftransport ToTransportMaltoseByPorinChannel
 :annotation (goid "GO:0015481")
 :across go/membrane
 :system FacilitatedDiffusion
 :via (owl-and Channel
          (owl-some (owl-some hasChannelType Porins))
 :cargo (owl-and ch/maltose
                (owl-some hasDaSize [(span < 1000)])
                   (owl-some hasConcentration
                      HighConcentration)))
```

Listing 6.14: The transport of the molecule maltose through a porin.

Calcium $Ca2^{+}$-activated potassium K^{+} channels are another special types of channels that are stimulated by the internal increase of calcium ion concentration gradient, which enable the passage of potassium ions across a lipid bilayer down a concentration gradient [109]. The three channels have been categorised into small-conductance(SK) intermediate-conductance (IK) and large-conductance (BK). The classes which describe the facilitated diffusion of potassium through the three $Ca2^{+}$-activated channels is defined in Listings 6.15 to 6.17.

```
(deftransport ToTransportPotassiumIonBySmallConductance
   ChannelCalciumActivated
 :annotation (goid "GO:0016286")
 :across go/membrane
 :system FacilitatedDiffusion
 :via (owl-and Channel
         (owl-some hasStimulus
          (owl-and ch/calcium_2+_
           (owl-some hasConcentration HighConcentration)
           (owl-some occurs_in go/intracellular)))
         (owl-some hasKCaTypeChannel small-conductance))
 :cargo (owl-and ch/potassium_1+_ (owl-some hasConcentration
    HighConcentration)))
```

Listing 6.15: The transport of potassium K^+ using small conductance channel.

```
(deftransport
   ToTransportPotassiumIonByIntermediateConductanceChannel
   CalciumActivated
 :annotation (goid "GO:0022894")
 :across go/membrane
 :system FacilitatedDiffusion
 :via (owl-and Channel
          (owl-some hasStimulus
           (owl-and ch/calcium_2+_
             (owl-some hasConcentration HighConcentration)
             (owl-some occurs_in go/intracellular)))
          (owl-some hasKCaTypeChannel
            intermediate-conductance))
 :cargo (owl-and ch/potassium_1+_ (owl-some hasConcentration
    HighConcentration)))
```

Listing 6.16: The transport of potassium K^+ using intermediate conductance channel.

```
(deftransport
   ToTransportPotassiumIonByLargeConductanceChannel
   CalciumActivated
 :annotation (goid "GO:0060072")
 :across go/membrane
 :system FacilitatedDiffusion
 :via (owl-and Channel
          (owl-some hasStimulus
           (owl-and ch/calcium_2+_
            (owl-some hasConcentration HighConcentration)
            (owl-some occurs_in go/intracellular)))
          (owl-some hasKCaTypeChannel large-conductance))
 :cargo (owl-and ch/potassium_1+_ (owl-some hasConcentration
    HighConcentration)))
```

Listing 6.17: The transport of potassium K^+ using large conductance channel.

6.6.2.1 Uniporter activity

This is the transport of molecules down their concentration gradients across the cell membrane via a facilitated diffusion transport system. In the uniport process, integral membrane proteins do not require coupling energy for a solutes to move from a high concentration to a lower concentration, These proteins are classified as facilitated diffusion transporters; see Listing 6.18 for `hexose uniporter activity (GO:0008516)` definition.

```
(deftransport ToTransportHexoseByFacilitatedDiffusionCarrier
  :annotation (goid "GO:0008516")
  :across go/membrane
  :system FacilitatedDiffusion
  :via Carrier
  :cargo (owl-and ch/hexose (owl-some hasConcentration
     HighConcentration)))
```

Listing 6.18: Hexose uniporter activity (GO:0008516).

6.6.2.2 Lipid transfer activity

A group of TA classes that describe the inter-membrane transfer of lipids via lipid transfer proteins (LTPs) that are responsible for transporting solutes, such as *fatty acids*, *phospholipids*, *glycolipids* or *sterol* from one region of a membrane to a different region on the same membrane. For instance, the class of `phospholipid transfer activity (GO:0120014)`, describes the transfer of a phospholipid from the leaflet of a donor membrane using a specific carrier protein, which provides a

hydrophobic environment for the phospholipid and delivers it to the leaflet of an acceptor membrane. The definition of the class is shown in Listing 6.19.

```
(deftransport
   ToTransportPhospholipidByFacilitatedDiffusionCarrier
  :annotation (goid "GO:0120014")
  :system FacilitatedDiffusion
  :via Carrier
  :from go/leaflet_of_membrane_bilayer
  :to go/leaflet_of_membrane_bilayer
  :cargo ch/phospholipid)
```

Listing 6.19: phospholipid transfer activity.

6.6.2.3 High- and low-affinity transporters

High-affinity transporter proteins are reported to function only when the electrochemical gradient for a transported solute is at a low level in the extracellular region. For a substance to move against its concentration gradient, primary or secondary energy is required. Conversely, low-affinity transporters can only bind to a solute that has a high concentration gradient in the external environment, importing it into the cell [17]. For a substance to move along its concentration gradient, a facilitated diffusion transporter is required. The reason for introducing these types of membrane-embedded transporter proteins in this section is that there are several TA classes that describe the transport function of solutes using high- or low-affinity transporters. For instance, consider the definition of the class of `high-affinity oligopeptide transmembrane transporter activity (GO:0015334)`, in Definition(14). However, in GO, there is a limited representation of the transport system (i.e., active or facilitated diffusion) by which high- and low-affinity transporters function. According to [97], the "high-affinity transporter" and low-affinity transporter" terms do not indicate the transport system. One high affinity transporter can mediate an active transport process by exploiting the energy from an ion gradient, whereas, another high-affinity transporter can facilitate the movement of a solute through a channel. For example, in *Neurospora crassa* cells, a high-affinity potassium K^+ uptake process occurs via the active K^+-H^+ potassium-proton symporter [95] [96]. On the other hand, a high-affinity channel can mediate the transport of K^+ [20].

Definition: 14. *Enables the transfer of oligopeptide from one side of a membrane*

to the other. In high-affinity transport the transporter is able to bind the solute even if it is only present at very low concentrations..
high-affinity oligopeptide transmembrane transporter activity (GO:0015334)

Due to the absence of a clear explanation regarding the mechanism associated with the classes that represent high and low-affinity transport functions, the classes are classified as general classification classes, unless a class description stated that it is an active. For example, the low affinity class `low-affinity glucose:proton symporter activity (GO:0005359)` has stated that the transport function driven by H^+ symporter, see class definition in Listing 6.20.

```
(deftransport ToTransportLowAffinityGlucoseProtonSymporter
  :label "ToTransportLowAffinityGlucose:ProtonSymporter"
  :annotation (goid "GO:0005359")
  :across go/membrane
  :system Active
  :cargo (owl-and ch/glucose (owl-some hasConcentration
     LowConcentration))
  :driven (owl-and ch/proton (owl-some hasConcentration
     HighConcentration))
  :transports-with LowAffinity
  :direction SameDirection)
```

Listing 6.20: Low-affinity glucose:proton symporter activity (GO:0005359).

6.6.3 Energy-dependent transporters: active transport

The hierarchy of the active transport is divided between two main transport systems: primary active transport (see Section 6.3.1) and secondary active transports (see Section 6.3.2). The grouping class `active transmembrane transporter activity (GO:0022804)` in Listing 6.21 describes the movement of a molecular entity against their concentration gradient, either using a primary energy source or from the energy stored in a substance gradient.

```
(deftransport ToTransportSoluteByActiveTransmembrane
 :annotation (goid "GO:0022804")
 :across go/membrane
 :system Active
 :cargo (owl-and ch/chemical_entity
             (owl-some hasConcentration LowConcentration))
 :driven (owl-or PrimaryEnergySource
             (owl-and ch/chemical_entity
             (owl-some hasConcentration HighConcentration))))
```

Listing 6.21: Active transmembrane transporter definition.

6.6.3.1 Primary active transporters

The set of the primary energy sources listed in Listing 6.22 using the value partition pattern(see Section 3.3) are used by the primary active transporters to drive active transport of a solute. For example, in Listing 6.23 we show the definition of the `light-driven active transmembrane transporter activity (GO:0015454)` that describe the transport of a solute across a membrane, driven by light.

```
(defpartition PrimaryEnergySource
  [Light Decarboxylation Oxidoreduction
   MethylTransferReaction ATP_Hydrolysis
   Phosphoenolpyruvate]
  :super ValuePartition
  :comment "Active transporters use a primary source of
     energy to lead the active transport of a substance or a
     group of substances against a concentration gradient."
```

Listing 6.22: The primary energy sources that power the active transporters.

```
(deftransport ToTransportSoluteByPrimaryActiveTransmembrane
   DrivenByLight
  :annotation (goid "GO:0015454")
  :across go/membrane
  :system Active
  :cargo (owl-and ch/chemical_entity (owl-some
     hasConcentration LowConcentration))
  :driven Light)
```

Listing 6.23: Ligh-driven primary active transmembrane transporter definition.

The vast majority of the primary active transporter classes in TA hierarchy are related to the transporters that uses the energy released by the ATP hydrolysis process. These transporters have been defined within the grouping class `ATPase-coupled transmembrane transporter activity (GO:0042626)` with a total number of 136

classes. As a result, we developed a pattern that can generate the ATP-dependent transporter classes, the pattern is shown in Listing 6.24 and include the ATP Hydrolysis energy as the driven mechanism for the transporters in Listing 6.25.

```
(defn substance-transporting-ATP_Hydrolysis [lis]
`(deftransport ~(symbol (str "ToTransport" (first lis) "
   TransmembraneDrivenWithATPase"))
 :annotation (goid ~(second lis))
 :across go/membrane
 :system Active
 :driven ATP_Hydrolysis
 :cargo
  ~(cond
     (= 3 (count lis))
       `(owl-and ~(nth lis 2) (owl-some hasConcentration
          LowConcentration))
     (= 5 (count lis))
       `(owl-and ~(nth lis 2) (owl-some hasConcentration
          LowConcentration)
          (owl-some ~(nth lis 3) ~(nth lis 4)))
     (= 6 (count lis))
       `(owl-and ~(nth lis 2) (owl-some hasConcentration
          LowConcentration)
          (owl-some ~(nth lis 3) ~(nth lis 4) ~(nth lis 5)))
     (= 7 (count lis))
       `(owl-and ~(nth lis 2) (owl-some hasConcentration
          LowConcentration)
          (owl-some ~(nth lis 3) ~(nth lis 4))
          (owl-some ~(nth lis 5) ~(nth lis 6)))
     (= 8 (count lis))
       `(owl-and ~(nth lis 2) (owl-some hasConcentration
          LowConcentration)
           (owl-some ~(nth lis 3) ~(nth lis 4))
           (owl-some ~(nth lis 5) ~(nth lis 6) ~(nth lis 7)))

  :else (println "Class data out of the range"))))

;; macro function to do the classes mapping
(defmacro deftransporters-driven-by-ATP_Hydrolysis [& lis]
  `(do ~@(map substance-transporting-ATP_Hydrolysis lis)))
```

Listing 6.24: ATP-dependent active transporters pattern.

```
(deftransporters-driven-by-ATP_Hydrolysis
  ["Substance"  "GO:0042626" ch/chemical_entity]
  ["Thiamine"  "GO:0048502"  ch/thiamine]
  ["Beta-glucan""GO:0015441" ch/beta-D-glucan]
  ["Ion"    "GO:0042625"     ch/ion]
  ["Cation" "GO:0019829"     ch/cation]
  ["Ferric" "GO:0015408"     ch/iron_3+_]
  ["Copper"  "GO:0043682"    ch/copper_2+_]
  ["Taurine" "GO:0015411"    ch/taurine  has-biological-role
     ch/xenobiotic]
  ....
  )
```

Listing 6.25: ATP-dependent active transporter classes.

In the case that an ATP-dependent class description specificities the starting cellular location of the transport process and the end position. They will be defined in the sub-pattern; that is quite similar to the original pattern, except it contains the `:from` and `:to` object properties. For example, `L-arabinose-importing ATPase activity (GO:0015612)`, indicates that the direction of the *L-arabinose* transport is from the external side of a cell membrane to the internal side. However, not all ATP-dependent classes followed the designed patterns. Exceptions that have been defined individually, such as the `ATPase coupled ion transmembrane transporter activity involved in the regulation of presynaptic membrane potential (GO:0099521)` (see the class definition in Listing 6.26).

```
(deftransport ToTransportIonTransmembraneDrivenWithATPase
   InvolvedInRegulationOfPresynapticMembranePotential
  :annotation (goid "GO:0099521")
  :comment "Involved in regulation of presynaptic membrane
    potential"
  :across go/membrane
  :system Active
  :driven ATP_Hydrolysis
  :occurs go/presynaptic_membrane
  :involved go/regulation_of_presynaptic_membrane_potential
  :cargo (owl-and ch/ion (owl-some hasConcentration
    LowConcentration)))
```

Listing 6.26: ATP-dependent ion transmembrane transporter occurs in the presynaptic membrane.

The energy generated from the ATP hydrolysis process does not only drive solutes

across the membrane of a cell, but also within and between intracellular membranes. There are a number of classes describe the transport of lipids from one membrane leaflet to the other (i.e., within a membrane region) via *flippases* and *floppases* transporters that utilise the ATP hydrolysis energy. Consider the definition of the class `phosphatidylserine flippase activity (GO:0140346)` in Definition(15) and itis implementation in Listing 6.27).

Definition: 15. *Catalysis of the movement of phosphatidylserine from the exoplasmic to the cytosolic leaftlet of a membrane, using energy from the hydrolysis of ATP.* *phosphatidylserine flippase activity (GO:0140346)*

```
(deftransport ToTransportPhosphatidylserineIntramembrane
   FlippaseDrivenWithATPase
 :annotation (goid "GO:0140346")
 :system Active
 :driven ATP_Hydrolysis
 :from go/ectoplasm
 :to go/cytoplasmic_side_of_endosome_membrane
 :cargo (owl-and ch/phosphatidyl-L-serine
                 (owl-some hasConcentration LowConcentration)))
```

Listing 6.27: Phosphatidylserine flippase activity (GO:0140346).

6.6.3.2 Secondary active transporters

These the energy held in the electrochemical gradient of a molecule or ion to transfer other molecules against their concentration gradients and across cell membranes. Besides the TA representation of secondary active classes and external related databases, we relied on the significant effort that has been made by Milton Saier [98] to build and improve active transporters, including secondary active transporters. Milton Saier's paper provides a detailed classification of transmembrane molecular transport systems, including transported substances, primary-energy and energy-coupling sources, transport systems, channel types and several biological and biochemical aspects[3]. According to Milton Saier's paper, secondary active transporters are driven by either ions' or solutes' electrochemical gradients. The secondary active pattern in Listing 6.28) is different to the earlier designed patterns as it must include the driving ion or molecule and the direction of the transport process (i.e., symporter or antiporter).

[3]The characterised transporters are currently available at the Transporter Classification Database (TCDB) `http://www.tcdb.org`

```
(defn secondary-substance-transporters [lis]
`(deftransport ~(symbol (str "ToTransport" (first lis) "
   BySecondaryActiveTransport"))
  :annotation (goid ~(second lis))
  :across go/membrane
  :system Active
  :cargo
  ~(cond
     (= 3 (count lis))
      `(owl-and ~(nth lis 2) (owl-some hasConcentration
         LowConcentration))
     (= 5 (count lis))
      `(owl-and ~(nth lis 2) (owl-some hasConcentration
         LowConcentration)
          (owl-some ~(nth lis 3) ~(nth lis 4)))
     (= 6 (count lis))
      `(owl-and ~(nth lis 2) (owl-some hasConcentration
         LowConcentration)
         (owl-some ~(nth lis 3) ~(nth lis 4) ~(nth lis 5)))
     (= 7 (count lis))
      `(owl-and ~(nth lis 2) (owl-some hasConcentration
         LowConcentration)
            (owl-some ~(nth lis 3) ~(nth lis 4))
            (owl-some ~(nth lis 5) ~(nth lis 6)))
     (= 8 (count lis))
      `(owl-and ~(nth lis 2) (owl-some hasConcentration
         LowConcentration)
          (owl-some ~(nth lis 3) ~(nth lis 4))
          (owl-some ~(nth lis 5) ~(nth lis 6) ~(nth lis 7)))
    :else (println "Class data out of the range"))))
  :driven (owl-and (owl-or ch/ion ch/chemical_entity)
                (owl-some hasConcentration HighConcentration)
                   )
  :direction (owl-or SameDirection OppositeDirection)))
```

Listing 6.28: Secondary active transporters.

One limitation of the GO representation of the symporter and antiporter classes is that it does not make a clear distinction between the solute being transported and the driving ion (discussed earlier; see Section 6.3.2). During our development of the symporter and antiporter classes, we refer to Table 11 in Milton Saier's paper and the TCDB (`www.tcdb.org`) to identify the solutes that are transported and their energy source, represented in another solute. In addition, we investigate related external databases such as Rhea, EC, KEGG, Reactome and MetaCyc, as well as making reference(s) to the source of the information associated with each GO class.

However, in some situations, the source of energy for a symporter or an antiporter has been stated to be unknown or can be one of many possible solutes, such as an ion (e.g., Na^+, K^+, H^+) or other solutes. The patterns (see Listing 6.29) that generate the symporter and antiporter classes are given as two solutes: the first value in the list is the driven solute and the second is the driving solute. Then, there are the GO IDs and the equivalent classes from the ChEBI ontology for the driven and driving solutes, see Figure 6.6.

By drawing such distinctions, we are increasing the ontology expressivity and answering questions that are hard to answer with the current version of GO, such as: what solutes are driven by the Na^+ symporter?

```
(defn secondary-substance-symporter [lis]
`(deftransport ~(symbol (str "ToTransport" (first lis) ":"(
   second lis)"Symporter"))
  :annotation (goid ~(nth lis 2))
  :across go/membrane
  :system Active
  :driven (owl-and ~(nth lis 3)
     (owl-some hasConcentration HighConcentration))
  :cargo
  ~(cond
     (= 5 (count lis))
      `(owl-and ~(nth lis 4) (owl-some hasConcentration
         LowConcentration))
     (= 7 (count lis))
      `(owl-and ~(nth lis 4) (owl-some hasConcentration
         LowConcentration)
               (owl-some ~(nth lis 5) ~(nth lis 6)))
    (= 8 (count lis))
     `(owl-and ~(nth lis 4) (owl-some hasConcentration
        LowConcentration)
           (owl-some ~(nth lis 5) ~(nth lis 6) ~(nth lis 7)))
   (= 9 (count lis))
     `(owl-and ~(nth lis 4) (owl-some hasConcentration
        LowConcentration)
              (owl-some ~(nth lis 5) ~(nth lis 6))
              (owl-some ~(nth lis 7) ~(nth lis 8)))
    :else (println "Class data out of the range"))))
 :direction SameDirection))

;; macro function to do the classes mapping
(defmacro defsecondary-substance-symporter [& lis]
  `(do ~@(map secondary-substance-symporter lis)))
```

Listing 6.29: Symporter Secondary active transporters.

```
(defsecondary-substance-symporter
;;["Driven Solute"-- "Driving Solute"--"GO ID" ---- "Driving solute from Chebi"---"Driven solute from Chebi"]
  ["Solute"          "Ion"          "GO:0015293"       ch/ion                    ch/molecular_entity]
  ["Solute"          "Cation"       "GO:0015294"       ch/inorganic_cation       ch/molecular_entity]
  ["Acetate"         "Cation"       "GO:0043893"       ch/inorganic_cation       ch/acetate]
  ["Anion"           "Cation"       "GO:0015296"       ch/inorganic_cation       ch/inorganic_anion]
  ["Chloride"        "Cation"       "GO:0015377"       ch/inorganic_cation       ch/chloride]
  ["Chloride"        "Potassium"    "GO:0015379"       ch/potassium_1+_          ch/chloride]
  ["Carbohydrate"    "Cation"       "GO:0005402"       ch/inorganic_cation       ch/carbohydrate]
  ["Melibiose"       "Cation"       "GO:0015487"       ch/inorganic_cation       ch/melibiose]
  ["Glucuronate"     "Cation"       "GO:0015488"       ch/inorganic_cation       ch/glucuronate]
  ["Sucrose"         "Cation"       "GO:0009669"       ch/inorganic_cation       ch/sucrose]
  ["AminoAcid"       "Cation"       "GO:0005416"       ch/inorganic_cation       ch/amino_acid]
  ["Nucleobase"      "Cation"       "GO:0015391"       ch/inorganic_cation       ch/nucleobase]
  ["Uracil"          "Cation"       "GO:0015505"       ch/inorganic_cation       ch/uracil  has-application-role ch/drug]
  ["Hexuronate"      "Cation"       "GO:0015539"       ch/inorganic_cation       ch/hexuronate]
  ["SialicAcid"      "Cation"       "GO:0015306"       ch/inorganic_cation       ch/N-acetylneuraminic_acid has-biological-
  ["AminoAcid"       "Potassium"    "GO:0017032"       ch/potassium_1+_          ch/amino_acid]
  ["Solute"          "Bicarbonate"  "GO:0140410"      ch/hydrogencarbonate       ch/molecular_entity]
  ["zinc"            "Bicarbonate"  "GO:0140412"      ch/hydrogencarbonate       ch/zinc_cation]
  )
```

Figure 6.6: A screenshot of the symporter Secondary active transporter classes, the complete implementation available at https://github.com/phillord/hyper-go/blob/master/src/hyper_go/TA/active_transporter.clj

Most of the symporter classes are driven either by the concentrations of sodium or proton ions, therefore, we designed patterns that are specific to the transporters that use sodium and proton electrochemical gradients for energy for the transport process. There are some classes that could not be defined through the sodium or proton patterns. These classes together do not form a pattern, but their molecular functions can be summarised in the following:

1. Transporters utilise a single ion electrochemical gradient to transport multiple solutes with zero, one or multiple properties in a single transport process. An example of such a transporter is the class of `sodium:potassium:chloride symporter activity (GO:0008511)`, which is defined in Listing 6.30. The transporter exploits sodium (Na^+) electrochemical gradient to drive the transport of potassium (K^+), and chloride (Cl^-).

2. The opposite of (1), utilising multiple ions' electrochemical gradients to drive a single solute against its concentration gradient; see Listing 6.31.

3. High-affinity and low-affinity active transporters, see Listing 6.32.

4. A symporter or antiporter contributes to the regulation of a qualitative or quantitative trait of a biological quality; see an antiporter example in Listing 6.33.

```
(deftransport ToTransportChloride:Potassium:SodiumSymporter
 :annotation (goid "GO:0008511")
 :across go/membrane
 :system Active
 :cargo (owl-and ch/potassium_1+_
                 (owl-some hasConcentration LowConcentration))
        (owl-and ch/chloride
                 (owl-some hasConcentration LowConcentration))
 :driven (owl-and ch/sodium_1+_
                 (owl-some hasConcentration HighConcentration))
  :direction SameDirection)
```

Listing 6.30: sodium:potassium:chloride symporter activity (GO:0008511).

```
(deftransport ToTransportDopamine:Chloride:SodiumSymporter
 :annotation (goid "GO:0005330")
 :across go/membrane
 :system Active
 :cargo (owl-and ch/dopamine
                 (owl-some hasConcentration LowConcentration)
                 (owl-some has-application-role ch/drug))
 :driven (owl-and ch/sodium_1+_
                 (owl-some hasConcentration HighConcentration))
        (owl-and ch/chloride
                 (owl-some hasConcentration HighConcentration))
  :direction SameDirection)
```

Listing 6.31: dopamine:sodium:chloride symporter activity (GO:0005330).

```
(deftransport ToTransportLowAffinityGlucose:SodiumSymporter
 :annotation (goid "GO:0005362")
 :across go/membrane
 :system Active
 :cargo (owl-and ch/glucose (owl-some hasConcentration
    LowConcentration))
 :driven (owl-and ch/sodium_1+_ (owl-some hasConcentration
    HighConcentration))
 :transports-with LowAffinity
 :direction SameDirection)
```

Listing 6.32: low-affinity glucose:sodium symporter activity (GO:0005362).

```
(deftransport
   ToTransportIonAntiporterInvolvedInRegulationOfpresynaptic
   MembranePotentialAntiporter
 :annotation (goid "GO:0099520")
 :comment "Involved in regulation of presynaptic membrane
    potential"
 :across go/membrane
 :system Active
 :occurs go/presynaptic_membrane
 :involved go/regulation_of_presynaptic_membrane_potential
 :cargo (owl-and ch/ion (owl-some hasConcentration
    LowConcentration))
 :driven (owl-and ch/ion (owl-some hasConcentration
    HighConcentration))
 :direction OppositeDirection)
```

Listing 6.33: Ion antiporter activity involved in regulation of presynaptic membrane potential (GO:0099520).

6.6.3.3 Phosphotransferase System

Phosphotransferase System (PTS) is a carbohydrate transport system occurs in bacterial cells to allow the uptake of sugars by utilising the phosphate extracted from the high-energy molecule *phosphoenolpyruvate (PEP)*. The phosphorylation process requires the existence of cysteine or histidine residues, which serves as an acceptor for the transferred phosphoryl group in the targeted carrier proteins (e.g., HPr (histidine containing protein)). In addition, it involves several intracellular enzymes (Enzyme I), which assist in the phosphoryl group transfer process in order to eventually phosphorylate the sugar substrates upon entering the cell environment.

This type of transport is an active transport, but it is different from the primary and secondary active transport by which the transported solutes, such as *galactitol*, *fructose* and *glucose* are modified (i.e., phosphorylated) in a single biological process. For example, the class `protein-N(PI)-phosphohistidine-glucose phosphotransferase system transporter activity (GO:0022855)` describe the PEP-dependent transport of glucose across the cell membrane according to the reaction: `protein N-phosphohistidine + glucose(out) = protein histidine + glucose phosphate(in)`. The PTS is multifunctional because it provides energy for the transported sugars and involves enzyme-catalytic functions. Therefore, the GO representation of the PTS-dependent transporters are classified as `active trans-`

`porter` from the TA hierarchy and as `phosphotransferase activity, alcohol group as acceptor` from the *catalytic activity* hierarchy, see Figure 6.7.

At this point, our definition of the PTS-dependent transporter classes is only describing the transport process without specifying the involved enzymes. In Listing 6.34 we show the logical representation of the PEP-dependent transport of *glucose*. The substance *glucose* is translocated from a region outside a cell, across the membrane and modified to its phosphorylated form. The transport process driven by the energy from Phosphoenolpyruvate (PEP) that depends on the existence of cysteine or histidine residues [5].

```
(deftransport ToTransportGlucoseDrivenByPhosphohistidine
   Phosphoenolpyruvate
:annotation (goid "GO:0022855")
:across go/plasma_membrane
:cargo (owl-and ch/glucose
              (owl-some hasConcentration LowConcentration)
                (owl-some transports-from go/
                   extracellular_region )
                (owl-some transports-to go/intracellular))
        (owl-and ch/glucose_phosphate
                (owl-some transports-from go/intracellular)
                (owl-some transports-to go/intracellular))
 :driven (owl-and Phosphoenolpyruvate
                (owl-some dependent-on
                 ch/N_pros_-phospho-L-histidine_residue)))
```

Listing 6.34: protein-N(PI)-phosphohistidine-glucose phosphotransferase system transporter activity (GO:0022855).

6.7 Evaluation

The goal of the normalisation methodology and its extension hypernormalisation is to allow the creation of an explicit and modular ontology, with the aim of addressing several difficulties related to ontology development, such as reducing manual maintenance, reusing ontology classifications and facilitating ontology evolution. The steps to build a hypernormalised ontology – in our case, it is the transporter activity subontology – can be summarised as follows (explained in detail in Sections 2.5.2 and 2.5.3):

1. Disentangling the structure of the TA into two independent disjointed tax-

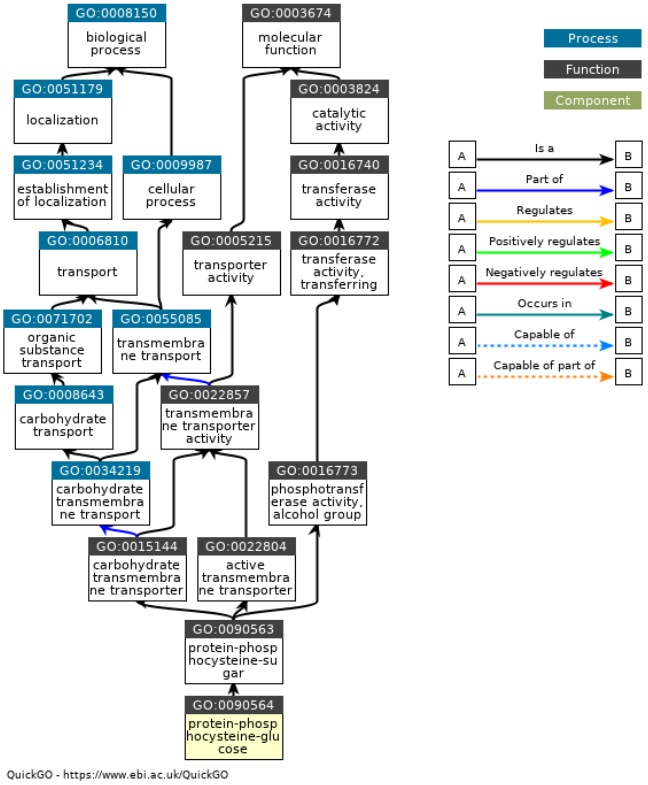


Figure 6.7: Example of a PTS-dependent class: the transport of glucose driven by phosphoenolpyruvate.

onomies: *self-standing classes* and *refining classes*.

2. The taxonomy of the *self-standing classes* includes only the core classes of the TA, which should be disjointed and most likely to be defined using *refining classes*.

3. The taxonomy of the refining classes includes the biological qualities and entities that are used to define self-standing classes, which are expected to be comprehensive, with possible overlapping between these classes in the entire refining hierarchy.

4. Relying heavily on designed patterns to build the self-standing classes and logical reasoners, to create the polyhierarchical relationships among them.

As the TA *self-standing classes* were defined in terms of the refining classes, we will begin by describing the hierarchical classification of the refining classes. In general, the hierarchy of the refining classes consists of three main parts that are independent of each other (see figure 6.8). .

- **Hyper-GO** refining-classes: The set of classes that we created to represent the biological qualities of the self-standing classes.
- ChEBI ontology: The ChEBI-structured classification of molecular entities is used to determines relationships between the transport activity classes. The ChEBI ontology is not directly linked to the refining hierarchy, but is imported in the same development environment and its entire ontological classification is considered as part of the refining hierarchy.
- CC and BP ontologies: The CCO and BPO classifications are also utilised to determine subsumption relations.

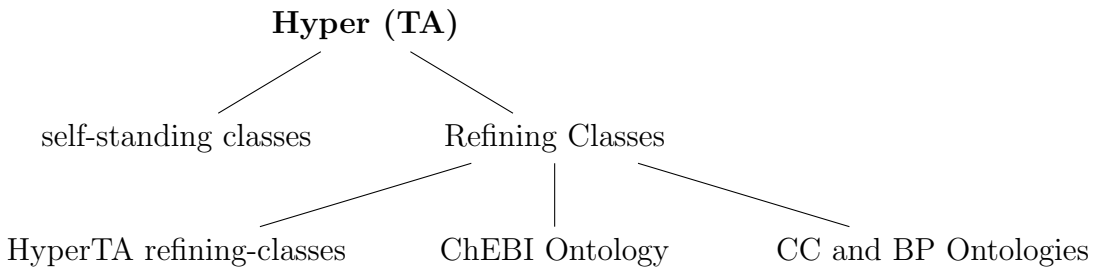


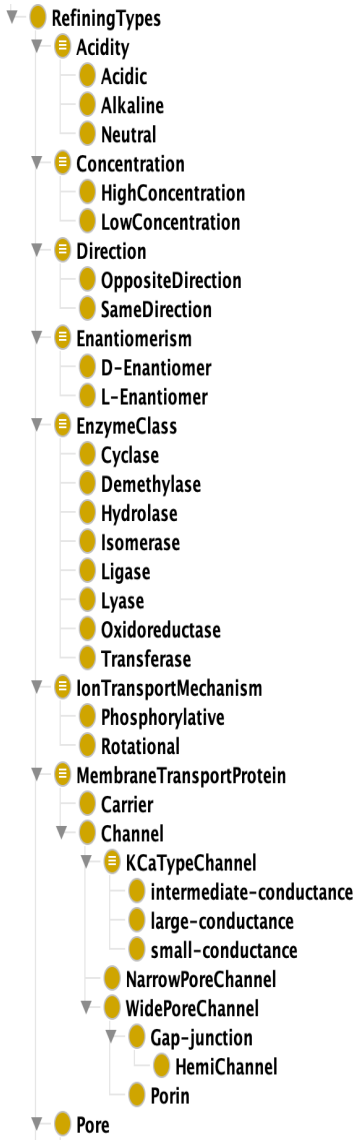
Figure 6.8: The general structure of the refining classes hierarchy.

The complete hierarchy of the **Hyper-GO** TA refining classes is shown in figure 6.9. Several elements have led to this hierarchy, starting with the original TA structure, the GO documentation, related external ontologies and databases, and commonly known biological references (mostly the work of Milton Saier [98]). We have used the higher-level patterns, such as the *value partition* and the *tier* (see Section 3.3) to build the refining properties in an easy and straightforward way. For example, consider the definition of the class Acidity in Listing 6.35, which has split into three categories: acidic, alkaline and neutral.

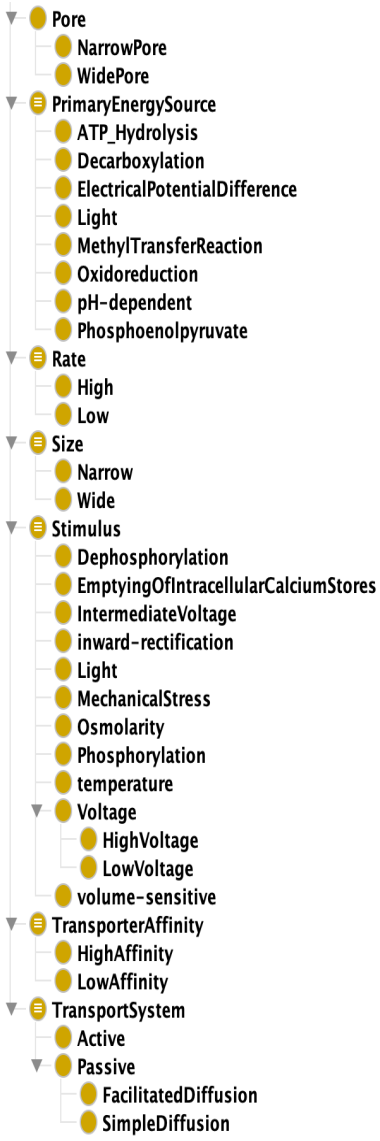
```
(defpartition Acidity
  [Acidic Neutral Alkaline]
  :comment "Amino acid, Basic, Acidic and Neutral amino-acid
     have different PH scale"
  :super RefiningTypes)
```

Listing 6.35: The tree of the Acidity class was generated from a single pattern.

According to the GO documentation [33], each activity (e.g. transport, catalysis or binding) represent a single function performed by a gene product, which may



(a) Refining classes hierarchy, part(1).



(b) Refining classes hierarchy, part(2).

Figure 6.9: The TA refining classes hierarchy

interact physically with other molecular entities to implement the task. Although a gene product may have several biological functions, a GO class should only represent a single function [4]. In transporter activity, each transport class describes a single transport function, and that function can either perform an active or passive transport based on the TA classification. During the development, there was no overlap between the classes of *active transport* and *passive transport*; similarly, the children of the *primary active* and *secondary active* transporters were disjointed.

In hypernormalisation, we do not assert subsumption relationships among the TA

[4]http://www-legacy.geneontology.org/GO.function.guidelines.shtml

self-standing classes. Instead, we exploit logical reasoners to build relationships among the classes using imported external ontologies and **Hyper-GO** refining classes and their ontological classification. In figure 6.10, we show part of the TA *self-standing classes* forming one level of hierarchy before reasoning. We have used the `HermiT OWL` reasoner (see Section 2.6.2) to infer the relationships among the hyper-transporter activity classes. The classes generally fall into one of the two distinct cellular transport systems: active or passive transport systems (see Figure 6.11).

Next, we will carry out an evaluation over the generated hyper transporter activity and compare the outcome with the unnormalised hierarchy in term of structure, statistics and maintainability.

6.7.1 structural evaluation

Initially, we thought that the hypernormalised polyhierarchical structure of the TA will be similar to the original unnormalised structure. However, there are a number of reasons, why they have not remained the same. The main reason is the difference in scope and semantics that the GO and ChEBI ontologies have. That is, the generality in the representation of the chemical entities that are involved in biological functions in GO versus the diversified and detailed description of the chemical entities in ChEBI (discussed in Section 5.3.4). Another reason is the new relationships have been inferred that were not created in the first place in the unnormalised hierarchy of the TA. For example, consider the class `glycine transmembrane transporter activity (GO:0015187)` that has been classified as a `is_a organic anion transmembrane transporter activity (GO:0008514)` and `is_a cation transmembrane transporter activity (GO:0008324)` in current TA representation. Conversely, in the hypernormalised version neither of the relationships exist, because in the ChEBI classification `glycine (CHEBI:15428) is_a organic acid (CHEBI:64709)`, but neither a `organic anion (CHEBI:25696)` nor `cation (CHEBI:36916)`. The reasoner inferred that the class `ToTransportGlycineTransmembrane (GO:0015187)` is subclass of the following classes:

1. ToTransportDrugTransmembrane
2. ToTransportL-AminoAcidTransmembrane
3. ToTransportNeurotransmitterTransmembrane

Figure 6.10: Part of the Hyper-TA classes prior to using a reasoner.

Figure 6.11: The Hyper-TA classes after running the HermiT OWL reasoner.

4. ToTransportNeutralAminoAcidTransmembrane

The reason for this difference is the mismatch in semantics between GO and ChEBI. Glycine as a substance is neither anion or cation, however, in solution which is its normal state biologically, it can potentially be either depending on the acidity of the solution. We have been unable to determine whether biologically, the state of Glycine while it is being transported, and it appears likely that GO has not either; as Glycine could be either an anion or a cation, so Glycine transporter activity could be either. ChEBI is more precise.

That led us to a question, how many classes that are classified as an `organic anion transmembrane transporter activity (GO:0008514)` and as a `cation transmembrane transporter activity (GO:0008324)` in the denormalised TA hierarchy and compare that with the hyper TA. There are 51 classes that have been classified as a direct subclasses of both `organic anion` and `cation` classes in the unnormalised hierarchy of TA, whereas in the hyper TA hierarchy there is no a single class that has been inferred to be an `organic anion` and `cation`. We used the "DL query tab" in the Protégé OWL editor to find out a class membership of other classes (e.g., subclasses, direct subclasses and superclasses).

Another example of a classification mismatch is the classification of the `methylammonium transmembrane transporter activity (GO:0015200)` class. In GO, the class is defined as an *is_a amine* and a *cation*, whereas in the hypernormalised ontology, the class is inferred only to be a *cation* (see Figure 6.12). As we defined the hypernormalised class using the equivalent ChEBI `methylammonium, (CHEBI:59338)` class, it appears that ChEBI defined *methylammonium* as an *ammonium ion* and not an *amine*. Conversely, the hypernormalised hierarchy of TA shows similar classifications to the original hierarchy. For instance, in Figure 6.13, the subclasses of the `amide transmembrane transporter activity` class are all inferred in the hypernormalised hierarchy as they were in the original hierarchy.

To summarise, the inferred structural representation of the Hyper-TA shows similar classifications to the original TA, but several differences as well. Due to the differences in scope and semantics that the GO and ChEBI ontologies have, not all of the original TA relationships have been inferred.

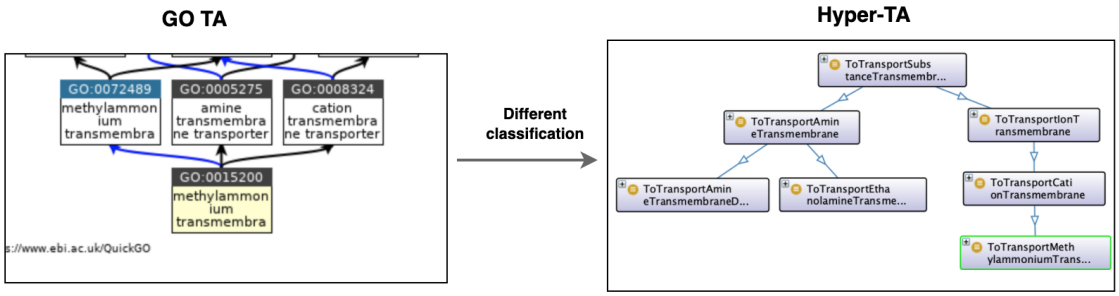


Figure 6.12: The Methylammonium transporter definition in GO TA versus Hyper-TA.

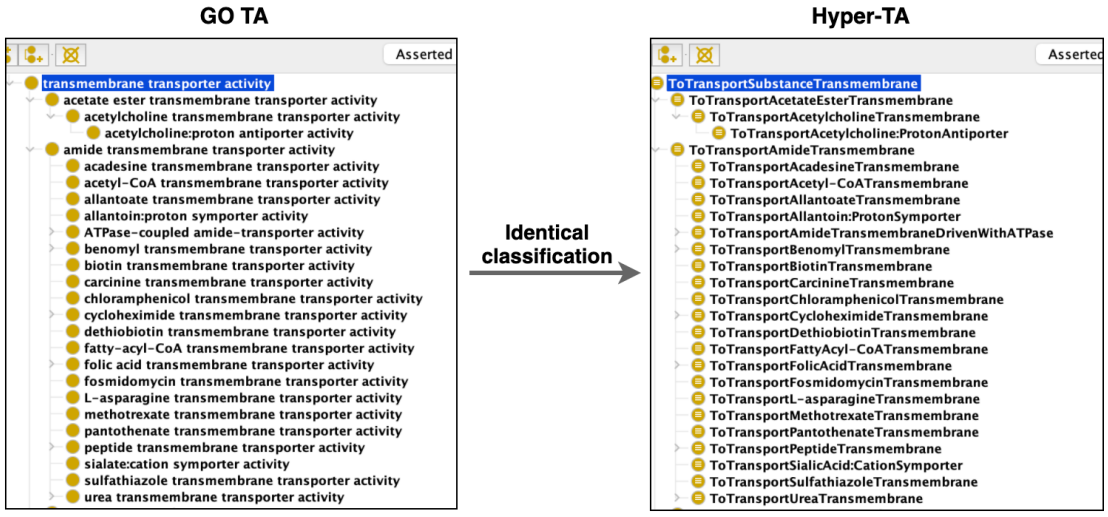


Figure 6.13: Part of the current GO TA classes classifications that were identically inferred in the Hyper-TA ontology

Hyper Ontology Metrics	total
Num of *self-standing classes*	1049
Num of *refining classes*	77
Num of used ChEBI classes	606
Total num of classes (before using a reasoner)	1735
Num of object properties	41
Num of data properties	1
Num of annotation properties	10
Num of classification	77
Num of inferred relations	1928

Table 6.3: The hyper TA ontology metrics.

6.7.2 Statistical evaluation

In this section we will highlight the hyper TA ontology statistics. In hypernormalised TA, the total number of the defined classes is 1125 class, which include both the *self-standing classes* and *refining classes*, but not the ChEBI classes. As mentioned earlier, the asserted subsumption relations only occur between the classes in the refining hierarchy. There are only 77 `SubClassOf` axioms in hyper TA, which define the asserted *is_a* relations. After the use of an ontology reasoner, the inferred relationships between the classes are 1928 subsumption relations. So only 4 percent of the hierarchical relationships have been explicitly defined, whereas the rest have been inferred using the reasoner. In table 6.3 we provide a summary of the hyper TA ontology statistics. The number of classes in the *self-standing classes* hierarchy and the *refining classes* hierarchy, and the number of ChEBI classes that are used in definitions. The number of classes in total and the number of object, data and annotation properties defined. Finally, the number of the asserted and inferred subsumption relationships.

The total number of classes in the denormalised TA hierarchy is around 1050 class, however, the number changes based on everyday changes made by the GOC ontology team and scientists. Currently (as of Nov 2020), there are 1899 subsumption relationships between the classes of `transporter activity (GO:0005215)`. The variation in the number of subsumption relationships between the GO TA and **Hyper-GO** TA is due to the different classification of chemical compounds in GO versus ChEBI and the daily changes on the TA hierarchy that have been made after

we have created the hyper TA.

6.7.3 Evaluation of maintenance and evolution

The aim of the hypernormalisation and patternisation techniques is to allow an ontology developer to manage the maintenance and evolution. Changes to GO classes occur on a daily basis, which mostly involves creating new classes or relationships and removing incorrect classes or inconsistent relationships. The addition or obsoletion of a class/relationship also requires checking multiple classifications, for example, moving the direct children of an obsoleted class to come under another class or multiple classes. By relying on the ChEBI high-quality classification of chemical terminologies and an automated reasoner, we overcome the issue of dealing with the complexity of the biological relations between the chemical entities.

Moreover, during the development of the Hyper-TA ontology, we aimed to create an ontology that could be evolved and extended smoothly, by designing ontological patterns to define the different cellular transporter classes. The pattern of `deftransport` is a *content-specific pattern* (see Section 2.6.1) that enables us to define the TA classes in a modular design by specifying the set of properties restricted to a transport function. A pattern-driven development approach eases the maintenance and evolution tasks by reducing the time needed to make the necessary changes. For instance, the class of `solute:bicarbonate symporter activity (GO:0140410)`, has been created recently. As we have designed the transporter-specific symporter pattern that can generate the symporter classes in Listing 6.29, we only needed to provide the class data from the definition in Listing 6.36 for the class to be created.

```
["zinc"  "Bicarbonate" "GO:0140412"  ch/hydrogencarbonate
   ch/zinc_cation]
```

Listing 6.36: solute:bicarbonate symporter activity (GO:0140410).

6.8 Discussion

In this chapter, we have described how we have utilised the hypernormalisation and patternisation approaches to rebuild `transporter activity (TA) (GO:0005215)`. TA is one of the broadest and most complex grouping classes as it covers many cellular transporter systems using a large number of ontology classes, constituting

approximately 10% of all classes in MFO. The generated hypernormalised TA is more explicit, modular and developed with the minimum amount of effort. The idea of hypernormalisation is to reduce the amount of knowledge that an ontology developer has to hold in their mind at any one time; they only need to have enough knowledge to build the list of definitions, and formal descriptions that allow an automated reasoner to determine the relationships between ontology classes.

More specifically, we have shown how we disentangled the structure of the TA ontology, dividing it into two independent disjointed taxonomies: *self-standing classes* and *refining classes*. Most of the *refining classes* are defined using the high-level patterns provided by [77] that allow the easy and accurate construction of hierarchies for classes and properties. This enables the utilised reasoner to infer the correct relationships among the *self-standing classes*, as they are defined, with respect to the classification of the *refining classes*. Because the TA classes define the functions of transporter proteins that move chemical entities across a cell environment, ChEBI is used. The ChEBI classification of chemical terminologies plays a vital role in rearranging the relationships in the TA hierarchy. As such, the final structure of the Hyper-TA is dependent on the *refining classes* and the ChEBI classifications.

In the current TA, the TA classes are distributed over three categories: general transport classification classes, *passive transport* classes and *active transport* classes. In general, the definitions and classifications of these classes and their children in GO were expressed clearly. There was a clear distinction made between the *active transport* classes and *passive transport* classes via disjointed hierarchies. Nevertheless, there is insufficient information on the cellular transport system by which high- and low-affinity transporters perform their functions. In high-affinity transport, the transporter is able to bind the solute only if it is present at low concentrations. For a substance to move against its concentration gradient, primary or secondary energy is required. While in low-affinity transport, the transporter is able to bind the solute only if it is present at very high concentrations. For a substance to move along its concentration gradient, a facilitated diffusion transporter is needed. There is evidence that the terms “high-affinity” and “low-affinity” are not associated with a single transport system (see Section 6.6.2.3). Conversely, a recent study carried out

by Dreyer and Michard [34] has indicated that the concept of "high- and low-affinity transport systems" has no scientific grounds and is not a correct characterisation of transporter proteins. In the study, different K^+ transport systems were examined via computer-aided dry laboratory experiments, proving that channel and co-transporter proteins showed the same affinity towards the potassium concentration. That means that the membrane proteins K^+ channel and K^+/H^+ co-transporter that have been characterised as "low-affinity" and "high-affinity", respectively, were shown in simulated experiments to be independent of the K^+ concentration gradients. Due to this lack of certainty, in the Hyper-TA, the classes that represent high- and low-affinity transporters are defined either as general classifications or *active transport*, dependent on their definitions in GO.

To summarise, we have shown that it is possible to represent GO transporter activity in a hypernormalised form. We believe that this form of classification is robust[5], easy to maintain and can be developed and extended smoothly. This does not mean that the Hyper-TA has no incorrect classifications, as the original TA, but we find the Hyper-TA to be less error-prone.

In the next chapter, we will investigate the use of the hypernormalisation and patternisation techniques by rebuilding the Catalytic Activity (CA).

[5]The classification is robust because it was built using higher-level patterns and logical reasoners

7

Hypernormalisation of Catalytic Activity

Contents

7.1 Introduction

`Catalytic activity (CA) (GO:0003824)` is the broadest grouping class as it includes the largest number of classes, with 7,029, constituting 63 percent of all classes in MFO. The aim of the CA classes is to represent the molecular functions related to the catalysis of biochemical reactions at physiological temperatures. A biochemical reaction is a process of transforming a molecule (known as reactant) to a different molecule (known as product), a process that is catalysed by enzymes. Enzymes are proteins that act as biological catalysts, speeding up the rates of all different types of reactions that take place inside our cells by reducing the activation energy of these reactions. Every enzyme works by binding a substrate (reacting molecule) to a special environment (called the active site) within the enzyme body, to stabilise not only the reacting molecules but also the activation energy in that particular reaction. In some reactions, an enzyme binds to one reactant molecule, which creates several products, whereas, in other reactions, multiple substrates react together to produce one larger molecule. According to the International Union of Biochemistry (IUB), enzymes are classified into different categories: *oxidoreductases*, *transferases*, *hydrolases*, *lyases*, *isomerases* and *ligases*. Each category of enzyme is further classified into classes of enzymes, each of which has a specific task to perform upon a set of molecules in a particular chemical reaction. For instance, *oxidase* and *dehydrogenase* are enzymes in the group of oxidoreductases enzymes that aim to catalyse oxidation-reduction (redox) reactions. In other words, they catalyse the transfer of electrons from one reacting molecule (known as an electron donor) to another molecule (the electron acceptor).

In the section that follows, we provide further details about the CA ontology's classifications and statistics.

7.2 CA classes' representation and statistics

In the hierarchy of CA, there are eight categories of enzymes represented as high-level or grouping classes (i.e., not functions themselves). In Figure 7.1, we show these enzymatic classes and the total number of subclasses for each individual cat-

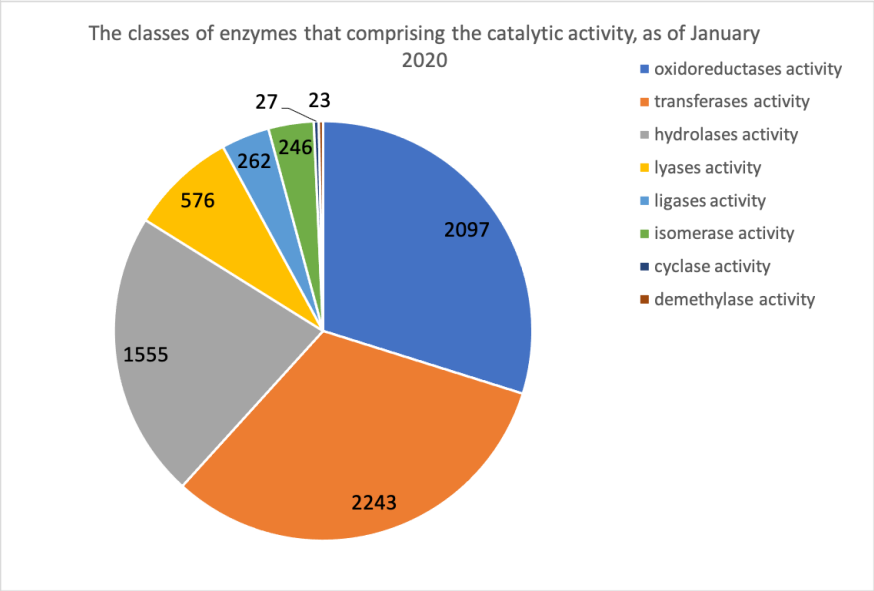


Figure 7.1: The classes of enzymes that comprising the catalytic activity, as of January 2020.

egory. The *transferases*, *oxidoreductases* and *hydrolases* categories of enzymes are the largest sub-hierarchies. In fact, each category of enzyme can be considered as a sub-ontology as they are almost entirely independent hierarchies. For instance, only four classes are subclasses of both the `oxidoreductase activity` and `transferase activity`, while no classes are shared between the hierarchies of `oxidoreductase activity` and `hydrolase activity`.

The majority of CA (also known as enzyme activity) classes describe chemical reactions using a chemical equation; this symbolic representation consists of reactant entities written on the left-hand side, product entities written on the right-hand side and a direction symbol to separate these (e.g., bidirectional (<=>) or left-to-right (=>)). In fact, almost all the biochemical reactions in the CA classes are defined with the = symbol, which means the net flux of a reaction is undefined. This form of definition is used as a standard definition for most of the CA classes; it is shown in Listings 7.1. Out of the total number of CA classes (i.e., 6,804 classes), 5,143 (75.5%) classes are defined using the formula in Listings 7.1. However, the number of substances before and after a chemical transformation process is variable and in some cases unknown.

```
Catalysis of the reaction:[reactants ↔ products]
```

Listing 7.1: Catalytic activity classes standard definition.

The ontology of CA also has "generic" reactions classes (known in GO as grouping classes) that do not represent functions themselves but are defined so as to group and classify classes with similar functions. These classes constitute approximately 24.4% of the total number of CA classes. For instance, consider the definition of the class `transferase activity, transferring alkyl or aryl (other than methyl) groups (GO:0016765)` in Definition(16).

Definition: 16. *Catalysis of the transfer of an alkyl or aryl (but not methyl) group from one compound (donor) to another (acceptor).*
transferase activity, transferring alkyl or aryl (other than methyl) groups, (GO:0016765).

In brief, the ontology of CA consists of grouping classes, with definitions that describe the overall function of an enzyme, and narrower functional classes that often represent the actual chemical reactions catalysed by the enzyme using a chemical equation.

7.3 Databases of chemical reactions

There are several online resources to describe enzymatic reactions, such as MetaCyc [23], KEGG [69], EC [39] and Rhea [4], with a different focus of each resource. Many of the GO CA classes have been cross-referenced to these databases and other resources. However, the GO guideline for ontology editors illustrates that the future goal is to automatically populate the definitions of GO enzymatic reactions using the reactions from Rhea. Rhea is an extensive resource of biochemical reactions, in which the reactions' participants are defined using ChEBI entities along with their chemical structures. Therefore, we have utilised Rhea reactions to facilitate the accurate creation of CA chemical reactions, with assistance from the mapping files, *rhea2go*[1], and by manually searching for yet unmapped classes using Rhea and other related databases.

There are several problems that prevent the automatic creation of CA classes using the GO-Rhea mapping file, *rhea2go*:

1. The GO-Rhea mapping file does not cover all of the CA classes.

[1]The *rhea2go* file is available at the GO official website `http://geneontology.org/docs/download-mappings/`

2. The grouping classes of CA do not have references to Rhea so they need to be handled separately.

3. The mapping file includes classes that are not only related to the CA, but to other GO classes.

4. We need to maintain the CA grouping classes' classifications, i.e., the set of reactions catalysed by the `hydrolases activity` need to be created separately to those for the `transferases activity` reactions.

5. The matches in the *rhea2go* file have the two following problems of:

   (a) One GO class mapped to many Rhea classes (e.g., GO:0120204 mapped to Rhea:60132 and Rhea:60136).

   (b) Many GO classes mapped to one Rhea class (e.g., GO:0102353 and GO:0062205 mapped to Rhea:33983).

In the section that follows, we present our strategy for disentangling CA structure and design ontology patterns, thus enabling us to define Hyper-CA classes.

7.4 Hyper-CA development strategy

As mentioned earlier, the CA hierarchy can be divided into several sub-hierarchies (i.e., based on the category of an enzyme) and these can be developed independently as they are almost independent. Therefore, in this study, we decided to start our implementation with the `transferase activity (GO:0016740)`, by retrieving all of its subclasses' labels, GO IDs and textual definitions. As the majority of the classes were chemical equation classes, we began the development work by separating them from their grouping classes. This allowed us to create the equation classes more rapidly and in a semi-automated way, especially for those that had references to Rhea. We defined each reaction class by replacing the reaction participants with their equivalent ChEBI chemical compound using the data from the GO-Rhea and Rhea-ChEBI mapping files. In the case that a CA class had no reference to Rhea, we investigated other cross-references (e.g., MetaCyc and KEGG) that mostly have links to Rhea, or we directly searched the Rhea database and replaced the reaction

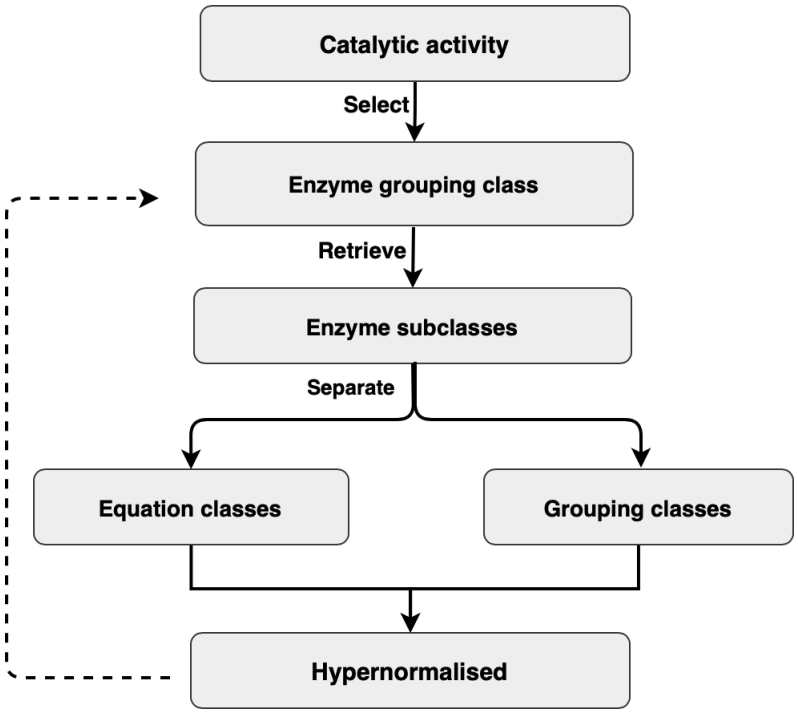


Figure 7.2: Overview of the CA classes development workflow.

participants with equivalent ChEBI compounds. Then, we relied on a reasoner to place classes automatically in the transferase hierarchy. In terms of the grouping classes, most of the `transferase activity` grouping classes' definitions describe the overall outcome of their equation subclasses using textual representation, and they often include meta-classes, that is, classes that do not map directly to chemical compounds in ChEBI. Therefore, we decided to start the development work with the chemical equation classes as generic classes require plentiful manual investigation and cannot necessarily be defined using logical definition if the reaction participants are not actual chemical compounds. An overview of the workflow used to create the CA classes is illustrated in Figure 7.2.

In Table 7.1, we provide an overview of the transferase activity classes, the number of generic classes, the chemical equation classes and the classes that also have relationships to other MFO activity, which are divided between `transporter activity` and `molecular transducer activity`.

Table 7.1: The classes comprising the transferase activity, as of January 2020.

Activity	No. of Generic Classes	No. of Equation Classes	No. of classes linked to other MF activity	Total
Transferase activity	328	1847	68	2243

Next, we will illustrate a number of challenges that need to be addressed related to the CA chemical equations classes development.

7.5 CA development challenges

1. Challenge: Reaction participants

The plan is for every chemical entity that participates in a CA chemical reaction to be replaced with its equivalent ChEBI compound using the mapping relations between GO and Rhea, and between Rhea and ChEBI. However, there are a number of difficulties associated with GO-ChEBI alignment:

1. Not all CA chemical reaction participants can be found in ChEBI. Missing chemical entities need to be submitted to the ChEBI web application and reviewed by ontology curators. Accepted requests are not directly available, but are made available in a monthly release update. We submitted more 30 pull requests to the ontology repository on GitHub to add new chemical entities.
2. "Generic" participants: GO and Rhea have generic compounds that are not represented in ChEBI.
3. Reaction participant names do not always match between GO and ChEBI; GO is more generic in the representation of chemical entities, whereas ChEBI is more specific and precise. Even within GO itself, the same entities appear with different representations, for example:
   - CO2 - carbon dioxide
   - FADH2 - FADH(2)
   - H2O - H(2)O

- H+ - H(+) - hydrogen
- NADP+ - NAD(P)+
- coumaroyl-CoA - 4-coumaroyl-CoA

2. Challenge: Reaction direction

There are four existing reaction directions: left-to-right direction(=>), right-to-left direction(<=), bidirectional (<=>) and undefined direction (=). The Gene Ontology Consortium (GOC) decided to use the non-directed (=) option for most of the classes where the net flux of a reaction was undefined. In the case of a reaction defined with a left-to-right direction or right-to-left direction, this can be represented relatively easily in the OWL format by defining object properties, for example – input and output (or reactant and product) to denote the direction. Conversely, the bidirectional and undefined direction implementations are much more complex as, in the case of an undirected reaction, we need to use the union-based approach to capture the meaning of the undefined reaction's direction (forward or backward). The representation of an undirected reaction is simplified in Listing 7.2.

```
(catalytic activity class
  (has reaction some
    and (hasReactant some ChEBI:X)
    and (hasReactant some ChEBI:Y)
    and (hasProduct some ChEBI:A)
    and (hasProduct some ChEBI:B)
  or
    and (hasReactant some ChEBI:A)
    and (hasReactant some ChEBI:B)
    and (hasProduct some ChEBI:X)
    and (hasProduct some ChEBI:Y))

and (hasEnzymeClass some Transferase))
```

Listing 7.2: A simplified representation of a reaction with undirected chemical reaction.

3. Challenge: Stoichiometry

A reaction stoichiometry is made up of the numerical relationships between reaction participants that are usually represented using numbers. This can be represented in the Web Ontology Language (OWL) format using a datatype

property (i.e., `owl:DatatypeProperty`). Alternatively, whenever the participants' amounts are determined in a CA class, we can repeat the definition of a reactant/product according to its quantity in a reaction. For instance, the definition of the CA class `all-trans-nonaprenyl-diphosphate synthase (GO:0052923)` indicates that the amounts of *isopentenyl diphosphate* and *diphosphate* molecules that are consumed and produced during the chemical reaction (see Definition(17)).

Definition: 17. *Catalysis of the reaction: geranyl diphosphate + 7 isopentenyl diphosphate = 7 diphosphate + all-trans-nonaprenyl diphosphate.*
all-trans-nonaprenyl-diphosphate synthase (geranyl-diphosphate specific) activity (GO:0052923)

7.6 Pattern-driven development

The main tool we have used to develop our ontology is the Tawny-OWL [76] library, which enables the construction of OWL ontologies and provides a set of patterns that explicitly support the creation of a hypernormalised ontology (see Chapter 3). We have extended the Tawny-OWL entities by defining a new entity, `defcatalyse`, to ease the creation of the CA classes. The `defcatalyse` entity is a new class with some property restrictions such as `hasReaction`, `hasReactant`, `hasProduct` and `hasEnzymeClass`. That is, we extend the basic Tawny-OWL frames with new frames that are restricted to the entity we developed, the `defcatalyse`. An example of a CA class created using the `defcatalyse` pattern is shown in Listing 7.3.

```
(defcatalyse ToCatalyseProteinSerineKinaseActivity
:annotation (goid "GO:0004674")
:annotation (database "RHEA:17989")
:reaction
(owl-or
 (owl-and
  (owl-some hasReactant ch/ATP_4-_ ch/L-serine_residue)
  (owl-some hasProduct ch/O-phospho-L-serine_2-__residue ch/
     ADP_3-_ ch/hydron))
 (owl-and
  (owl-some hasReactant ch/ADP_3-_ ch/
     O-phospho-L-serine_2-__residue ch/hydron)
  (owl-some hasProduct ch/ATP_4-_ch/L-serine_residue)))
:enzyme Transferase)
```

Listing 7.3: The definition of the `Protein serine kinase activity (GO:0004674)`.

On the other side, the "generic" reaction classes in the `transferase activity` hierarchy often describe the chemical transformation of functional groups (e.g., methyl group, acyl group or glycosyl group) from one substance (referred to as the donor) to another substance (refers to as the acceptor), but take a free-text format. As such, we found it challenging to change the generic definitions into logical definition patterns that allowed a logical reasoner to infer the relationships between these "generic" classes and their subclasses of equation classes. Moreover, in ChEBI, the classes of *groups* (e.g., `methyl group`, `glycosyl group` and so on) are defined in a separate hierarchy under the high-level class `group (CHEBI:24433)` and linked with their parent molecular entities using the relationship's *is substituent group from*, as stated in the ChEBI documentation. For example, in GO, the equation class, `methylamine-glutamate N-methyltransferase activity, (GO:0047148)` define the chemical equation shown in Definition 18. This was classified as a *SubClass Of* of the grouping class, `N-methyltransferase activity (GO:0008170)`, which describes the transfer of a *methyl group* from one molecule to another. Yet, in ChEBI, there is no relationship between the `methyl group (CHEBI:32875)`, and the molecular entity, `methylammonium`.

Definition: 18. *Catalysis of the reaction: L-glutamate + methylammonium = N-methyl-L-glutamate + NH(4)(+).*
methylamine-glutamate N-methyltransferase activity (GO:0047148)
Cross-References: RHEA:15837

Currently, there are two solutions to this modelling issue: one is to ignore the grouping classes, while another solution is to define them only with their GO definitions and IDs, and then manually assert the subsumption relations (i.e., *is-a-superclass-of...*) between "generic" reaction classes and equation classes, according to the GO classification. Although the latter solution requires a great deal of work and fails to observe the hypernormalisation [77] principle (i.e., to have a flatted hierarchy), it is used to maintain the GO classification of generic reactions with the Rhea-based equation reactions and the hierarchy is still normalised [93].

The development of grouping classes was divided into two steps: in the first step, we retrieved all the `transferase activity` grouping classes, with their names, definitions and IDs to be created using annotation properties; see an example in

Listing 7.4. In the second step, we asserted the relationships between every grouping class and its subclasses of equation classes, as demonstrated in Listing 7.5. This approach allowed for any future developments, such as changing the definitions of the grouping classes into logical definitions, as the subsumption relations were created separately.

```
(defclass ToCatalyseO-HydroxycinnamoyltransferaseActivity
 :annotation (goid "GO:0050737")
 :annotation (Def "Catalysis of the transfer of a
    hydroxycinnamoyl group to an oxygen atom on the
 acceptor molecule."))
```

Listing 7.4: The definition of the grouping class `O-hydroxycinnamoyltransferase activity (GO:0050737)`.

```
(as-subclasses
ToCatalyseO-HydroxycinnamoyltransferaseActivity
:disjoint
(declare-classes
ToCatalyseGlucarateO-HydroxycinnamoyltransferaseActivity
ToCatalyseGalactarateO-HydroxycinnamoyltransferaseActivity
ToCatalyseTartronateO-HydroxycinnamoyltransferaseActivity
ToCatalyseShikimateO-HydroxycinnamoyltransferaseActivity
ToCatalyseGlucarolactoneO-HydroxycinnamoyltransferaseActivity
ToCatalyseQuinateO-HydroxycinnamoyltransferaseActivity
ToCatalyseChlorogenate-GlucarateO-Hydroxycinnamoyltransferase
    Activity))
```

Listing 7.5: Manually asserting the relationship between the class `O-hydroxycinnamoyltransferase activity (GO:0050737)` and its subclasses.

7.7 Evaluation

In this section, we will carry out an evaluation of the generated classes of Hyper-CA by comparing the hierarchical relationships in the new hierarchy with those of the current GO CA hierarchy. This evaluation approach is known as "gold standard-based" for using the existing gold ontology (also known as the core ontology) to evaluate the target-related ontology. The comparison will mainly concern the relationships among the equation reaction classes, as the relationships between the "generic" reaction classes and the equation classes were asserted manually, as stated by GO. Conversely, the definitions of the CA equation reaction classes have changed into computable logical definitions that support the automatic inference of member-

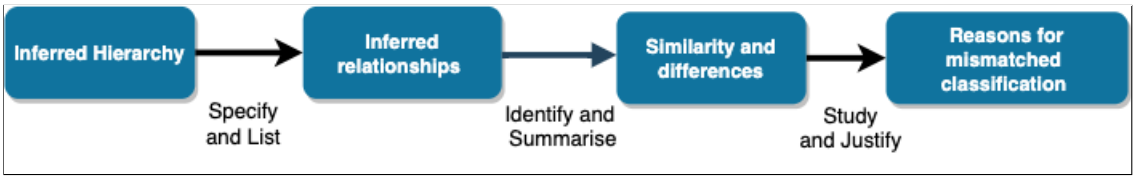


Figure 7.3: Overview of the workflow used to evaluate the Hyper-CA hierarchy.

ships by relying greatly on the ChEBI classification of chemical entities, with the assistance of Rhea as an intermediary. That means, the relationship between one equation class and another will be determined based on the relations between the chemical entities that participate in each reaction, according to their ChEBI classifications. In the implementation of logical definition patterns for the equation's chemical reactions, we aimed to design a logical definition that captured the meaning of undirected reactions, although there were simpler, logical definitions. In fact, the designed pattern for defining CA equation classes was derived from the discussions held by the developers of GO on the ontology repository on GitHUB [2]. There was a general consensus on the use of Rhea biochemical reactions, which we have exploited to populate the logical definitions of reactions using existing mappings (i.e., GO-to-Rhea and Rhea-to-ChEBI) and manual searching.

The aim of the hierarchical comparison was to check whether the relationships between the equation classes changed or remained the same after the use of logical definitions and equivalent Rhea biochemical reactions. The created classes only constitute 16% of the total number of CA classes due to the challenges mentioned earlier and the limited time available to create these classes. However, the rest of the CA classes are definable as they fall into the same identified categories: "generic" reaction classes or equation reaction classes. As we have not yet covered all of the CA classes, we are not able to assess the CA ontology as a whole. We performed the comparison manually by concentrating mainly on the inferred relationships. An overview of the workflow used to evaluate the generated hierarchy is presented in Figure 7.3.

Firstly, we specified all of the inferred relationships among the equation classes; then, we identified the similarities and differences between the hierarchies, before going

[2]https://github.com/geneontology/go-ontology/issues/14984

on to study the reasons for any differences. The total number of equation classes that we created using the pattern in Listing 7.3 is 760, all of which belong to the hierarchy of `transferase activity`. After the use of an OWL reasoner (specifically, the `HermiT OWL`) there were 30 inferred *SubClass Of* relationships. Out of the 30 inferred relationships, 22 (69%) subclass inferences matched the GO classification. The eight (28%) new relationships were generated based on the ChEBI relationships between the reaction's participants. For instance, the classes shown in Definitions 19, 20 and 21 are defined as siblings (having the same *superclass*) in current GO representations; see Figure 7.4. In **Hyper-GO**, new relationships were inferred to make the `glycine N-acyltransferase activity (GO:0047961)`, as a *superclass* for the other two classes. This is because the reaction participants `choloyl-CoA(4-)(CHEBI:57373)` and `benzoyl-CoA(4-)(CHEBI:57369)` in ChEBI are *SubClass Of* the class `acyl-CoA(4-)(CHEBI:58342)`. In addition, on the right hand side of the equation reactions the class of `N-acylglycinate(CHEBI:57670)` is the parent class for both `glycocholate(CHEBI:29746)` and `N-benzoylglycinate(CHEBI:606565)`.

Definition: 19. *Catalysis of the reaction: acyl-CoA + glycine = CoA + N-acylglycine.*
glycine N-acyltransferase activity (GO:0047961)
Cross-References: RHEA:19869

Definition: 20. *Catalysis of the reaction: benzoyl-CoA + glycine = N-benzoylglycine + CoA + H(+).*
glycine N-benzoyltransferase activity (GO:0047962)
Cross-References: RHEA:14001

Definition: 21. *Catalysis of the reaction: choloyl-CoA + glycine = CoA + glycocholate.*
glycine N-choloyltransferase activity (GO:0047963)
Cross-References: RHEA:18498

The result was discussed with the GO developers on the ontology GitHub repository[3]; the proposed relationships were checked and shown to be correct. The GO repository contains the source code of the ontology, which is edited only by its developers. Our plan is to compare the hyper CA with the denormalised CA once we reach a state of completion. In Table 7.2, we present the reminder of the new inferred relationships.

[3]https://github.com/geneontology/

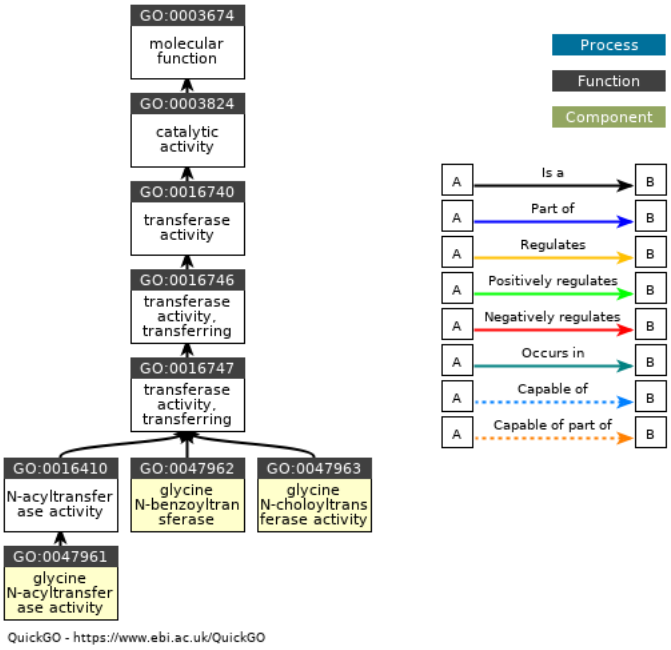


Figure 7.4: The GO classification of the CA classes glycine N-acyltransferase, glycine N-benzoyltransferase and glycine N-choloyltransferase.

Table 7.2: The difference between GO classification and the **Hyper-GO** classification.

No	In GO	**Hyper-GO**
1	GO:0003956 and GO:0030701 are siblings	GO:0030701 *SubClassOf* GO:0003956
2	GO:0047507, GO:0047506, GO:0036431 and GO:0004127 are siblings	GO:0047507 *SuperclassOf* the rest
3	GO:0047178 and GO:0047177 are siblings	GO:0047177 *SubClassOf* GO:0047178
4	GO:0050369 and GO:0034211 are siblings	GO:0034211 *SubClassOf* GO:0050369

Lastly, due to the large numbers of CA classes and the challenges mentioned earlier (see Section 7.5), we were unable to reformulate all the CA classes in a hypernormalised form. In addition, the project involved communication with external related parties (i.e., GO, ChEBI, and Rhea) to clarify unclear class descriptions, adding missing reactions participants, and reporting incorrect classifications. As a result, to redefine all CA classes, additional time and multiple team members are required. Once we reach a state of completion, the ideal evaluation process is to compare the completed Hyper-CA with the GO CA and related standardised databases of

chemical reactions, such as Rhea and KEGG.

7.8 Discussion

In GO, as it stands, the representation of the `Catalytic activity` classes is based on textual definitions and manually asserted relationships. In general, the ontology of CA consists of "generic" reaction classes, with definitions that describe the overall function of an enzyme, and more narrower functional classes that often represent the actual chemical reactions catalysed by the enzyme using a chemical equation. In this chapter, we have described the steps taken to change the representations of the CA classes into Rhea-based logical definition patterns. As the hierarchy of CA comprises several large sub-hierarchies, each of which represents a category of an enzyme, the initial step of the process involved selecting an enzyme class, disentangling its ontological structure and retrieving all of its subclasses. Then, separating the "generic" reaction classes from the equation reaction classes, and designing logical patterns that enabled the creation of classes and the automatic inferences of relationships. Lastly, we evaluated the newly generated hierarchy by making comparisons with the original hierarchy.

There have been many discussions about how to model and patternise the definitions of the equation classes. There is a need to address several challenges, such as the directionality of reactions and reactions' stoichiometries, and support the automatic inference of relationships. Within the GOC community, non-direction of reactions is the most adopted and used format, unlike with the Rhea database, which uses all four options for directionality. The logical patterns that we have designed for the non-direction of reactions require additional work, as more axioms must be defined and a union-based approach is used to capture the meaning of an undefined reaction's direction (forward or backward). However, this approach limits the number of classes to be defined by avoiding the need to create a class for each direction of a reaction, which would increase the size of the ontology. Of course, there are alternative computable patterns for defining equation classes, which are, in general, more simple than the union approach. For instance, a simple modelling solution to define an undirected reaction is shown in the example in Listing 7.6, using *has-reactant-left*,

has-reactant-right and *direction* properties.

```
(defcatalyse ToCatalyseProteinThreonineKinaseActivity
:annotation (goid "GO:0004674")
:annotation (database "RHEA:46608")
:reactant-left ch/ATP_4-_ ch/L-threonine_residue
:reactant-right  ch/ADP_3-_ ch/
   O-phosphonato-L-threonine_2-__residue ch/hydron
:direction undefined
:enzyme Transferase)
```

Listing 7.6: Alternative pattern to define the class `protein threonine kinase activity (GO:0004674)`.

It appears that OWL is not the best framework for representing chemical reactions, especially non-directed reactions. However, we contend that the logical definitions that we have designed are the best possible modelling solutions for defining chemical equations in OWL.

Lastly in this chapter, we have shown how we defined equation classes using logical patterns, but we were unable to change the definitions of the "generic" reactions into logical definition patterns. Therefore, we decided that the best decision would be to manually assert the relationships between equation reactions and "generic" reactions using the subsumption relations (i.e., *is-a-superclass-of*) from GO. These subsumption relations were created separately from the definitions of the "generic" reaction classes to ease, in the future, the development of any logical patterns.

In the next chapter, we will summarise the outcomes of this thesis and discuss how the use of the hypernormalisation and patternisation techniques affects the overall structure of the implemented part of GO.

8

Discussion

Contents

8.1 Introduction

The usage of ontologies in many scientific disciplines has increased to represent the variety and fast growth of scientific data. At present, the number of biomedical ontologies hosted in the NCBO BioPortal is more than 820, with nearly 10 million classes. Ontologies continue to grow in response to new scientific results and quality enhancements. However, as ontologies become larger and highly tangled, the need for initiatives to manage their scalability, maintainability and expressivity becomes essential. As a consequence, the number of methodologies and techniques has grown to address these concerns and other ontological engineering issues. Nevertheless, to our knowledge, there is no single standardised methodology that is widely used and covers all aspects of ontology development, regardless of the application domain.

The purpose of this research has been to address the scalability and ease of development of large ontologies by, firstly, suggesting a new identifier scheme for generating ontology identifiers, and secondly, investigating the hypernormalisation, pattern-driven development and programmatic approach by rebuilding the MFO.

The motivation for developing a new identifier scheme has been to facilitate scalability and overcome the concerns associated with some of the ontological standards, such as the use of numeric identifiers. In our scheme, Identitas, we have used random identifiers to enable concurrent development, and we exploited the *proquint* library to overcome problems with memorability and pronounceability. Moreover, we enabled the use of checksum to prevent the occurrence of errors while accessing relatively similar identifiers. In this chapter, we discuss the advantages, limitations and any possible improvements to this identifier scheme.

To overcome issues associated with ontology development and maintenance, we considered using the hypernormalisation approach. As a result, we have rebuilt the transporter activity hierarchy and some parts of the catalytic activity hierarchy using high-level patterns that explicitly support the creation of a hypernormalised ontology using the Tawny-OWL environment, with the assistance of related ontologies and logical reasoners. In this chapter, we discuss how this approach facilitated the development of the GO classes, along with the limitations of the approach and

possible improvements that could be taken into consideration.

In addition, while rebuilding the ontologies using the hypernormalisation approach, we were able to highlight inconsistent classifications and identify semantic mismatches between GO and ChEBI ontologies that have arisen as the ChEBI ontology has a higher level of specificity than GO. In this chapter, we discuss the outcomes of our investigation of the hypernormalisation technique.

8.2 The Identitas library

Ontology identifiers are the key for each entity defined in an ontology, and enable a unique and persistent reference to each term. There is clearly a wide consensus in favour of using *semantics-free* local IDs (identifiers that are unique within a database or an ontology), as an identifier that is based on some semantics associated with the term may need to be changed when that meaning changes. The style of the identifiers generated by most identification schemes either consists of numeric or alphanumeric characters, which introduces a number of problems for end-users. These can be, for instance, hard to read, memorise and pronounce, and it becomes easy to make mistakes. In addition, monotonically increasing numbers require a degree of coordination between ontology developers to obtain new IDs; this can significantly hamper or delay ontologies' development.

In Identitas, we enabled the concurrent development of ontologies by simply generating identifiers in a random manner, and utilised the *proquint* library to overcome problems with memorability and pronounceability. In addition, a checksum was implemented to prevent the occurrence of errors while accessing relatively similar identifiers. Finally, Identitas was integrated into two ontology development environments, Protégé and Tawny-OWL [76].

There are existing, accepted ways of overcoming the challenges with distributed development, such as using blocks of identifiers or centrally generating new IDs on-demand. Although these approaches are effective, they require developers to manage the IDspace accurately, since preallocated IDs cannot be used elsewhere. A centralised server requires developers to set up a connection to the server, therefore, it is sensitive to network issues or a lack of availability of the server. Decentralised

IDs such as UUIDs are unique worldwide but not checksummable.

Some limitations of the Identitas scheme should be noted. Firstly, we have not evaluated the readability of our IDs, nor has any prior researcher done this with proquints. We note, however, that Identitas IDs have three key advantages which are composable (i.e. they can be random or not, they can use checksums or not, they can use proquints or not). We did search extensively for an alternative scheme to *proquints*, but could not find one. Yet, it would be possible to incorporate alternatives within future versions of Identitas, perhaps similar to *what3words*. Secondly, we did not address the question of whether or not it is practical to move an existing ontology to this style of identifier. It might be hard work and the costs might be greater than benefits of moving ontologies to utilise Identitas. However, we have presented details of the scalability of Identitas, showing that it can easily scale to an ontology the size of GO. That is because GO IDs are assumed by many pieces of software, and porting GO IDs would be a significant effort. Finally, we did not give a practical evaluation. Clearly, the ideal test of Identitas would be to build a large ontology from scratch using Identitas; this is obviously impractical within the confines of this PhD research. Moreover, to our knowledge there has not been a formal evaluation of many existing identifier schemes, including those that have long been in use; in practice, once an identifier scheme has been established it is very often difficult to change. However, as a partial attempt at this form of evaluation, we have added Identitas IDs as a standard annotation property to the classes of our **Hyper-GO** ontology; this offers a migratory path for the use of Identitas without wholescale replacement of the existing scheme.

In summary, with Identitas, we have provided a number of features that address the scalability and ease development of ontologies (which fulfils RQ1). Moreover, we have discussed the limitations, and the future work required to improve Identitas.

8.3 Hypernormalisation of Transporter activity and Catalytic activity

In this thesis, we have utilised the hypernormalisation approach to rebuilding two hierarchies of transporter and catalytic activities, together constituting approximately

78% of all classes in the MFO. Firstly, acquired sufficient biological knowledge to understand the GO classification and determine the way in which we would apply the hypernormalisation technique. Although the MFO classes share attributes, they describe different molecular activities, such as transporting chemical entities or converting one entity to another. Therefore, we decided to apply our approach to each molecular activity independently, starting with transporter activity and moving on to catalytic activity.

In the process of hypernormalising the transporter activity (TA), we utilised the higher-level patterns described in [77], which assist with building explicit and accurate hierarchies of TA. Moreover, we designed *content-specific patterns* that aided in the classes' construction. We found that the TA classes are distributed over three categories: general transport classification classes, *passive transport* classes and *active transport* classes. However, there were challenges with defining more specific transporter classes that involved multiple biological/chemical entities and roles. For instance, there were difficulties with determining the types of high- and low-affinity transporters as there was insufficient information within the GO representation. The hyper-TA representation was based on the GO representation and related, reliable resources that are used by GO, such as the Transporter Classification Database [98].

Catalytic activity (CA) was the second part of the MFO that we investigated to apply our hypernormalisation approach. The investigation showed that the CA hierarchy consists of several large sub-hierarchies, each of which represents a category of an enzyme. As such, we found it easier to apply our approach by, firstly, selecting an enzyme class, then disentangling its structure, retrieving all of its subclasses and rebuilding them. In addition, we found that a CA class is either one of two types: a "generic" reaction class or chemical equation class. So, we designed a logical pattern that enables the definition of the non-direction equation reactions and supports the automatic inference of memberships, relying on the ChEBI classification of chemical entities. Cross-references between GO and Rhea facilitate the reactions' creation, as Rhea uses ChEBI classes to describe the chemical compounds that take part in the reactions. However, not all the CA classes have been referenced; therefore, we had to make an effort to find the equivalent entries from Rhea. See Chapter B for the

list of CA classes that we have mapped to their equivalent in Rhea[1]. Meanwhile, there were difficulties in changing the representations of "generic" reactions into logical definition patterns as the relationships between the "generic" classes and equation classes could not be inferred using the equivalent chemical entities from the ChEBI ontology. The "generic" classes describe the chemical transformation of functional groups (e.g., methyl group), in ChEBI the functional groups defined in a separate hierarchy and have no direct relationships with equation substances. We found that the best way to model these "generic" classes was to assert the relationships between the equation reactions and "generic" reactions manually, using the subsumption relations.

While rebuilding GO using the hypernormalisation approach, we identified semantic disagreement between GO and ChEBI. Although the two ontologies describe the same chemical entities, GO uses a class representing a concept that does not directly map to a concept in ChEBI. ChEBI provides a detailed representation of molecular entities; for instance, it distinguishes between the L-, D- and generalised forms of a molecule. GO, meanwhile, describes a biological situation, where all of the biological states of a chemical entity in question are in solution, where the distinction is less meaningful. We were unable to determine whether biologically, the state of a molecule while it is being involved in an activity, and it appears likely that GO has not either. Therefore, we decided to rely on the equivalence axioms created by the GO-Plus ontology.

In general, the advantages of applying the hypernormalisation can be summarised as following:

1. Reduce the amount of knowledge that an ontology developer has to hold in their mind at any one time; they only need to have enough knowledge to build the list of definitions and formal descriptions that allow an automated reasoner to determine the relationships between ontology classes.

2. Support the creation of ontological higher-level patterns, and recast existing patterns to build ontologies. Pattern-driven development is a well-known ap-

[1]Most of the mappings were found using related databases (e.g., MetaCyc and KEGG) using the Cross-references relation from GO.

proach for easing the maintenance and evolution tasks by reducing the time needed to make the necessary modifications.

3. Build compositional and large ontologies in an explicit, modular and maintainable form.

4. A hypernormalised ontology is built from several modules that can be re-used, maintained and developed independently with minimal effort.

8.3.1 Future Work

8.3.1.1 Practical Immediate Next Steps

For our future work, we seek to continue the development of the catalytic activity classes, evaluating and discussing any new inferred relationships. In addition, we wish to apply hypernormalisation to other molecular activities and compare the results with the original ontology. In terms of the part of GO that is hypernormalised, we could review it and improve the designed patterns in response to future developments of the GO classes. We believe that this research will be of interest to ontology developers, as we are showing that it is possible to reconstruct an ontology using the hypernormalisation approach.

In summary, this investigation of the hypernormalisation methodology has resulted in the development of a hypernormalised form of the transporter and catalytic activity hierarchies (this fulfils RQ2). Employing our hypernormalisation methodology has resulted in the construction of explicit, manageable and robust hierarchies (this fulfils RQ2.1). Moreover, it has allowed us to highlight inconsistent classifications and identify semantic mismatch between GO and ChEBI ontologies (this fulfils RQ2.2). Our ability to query the generated ontology increased expressively after using computed logical definitions (this fulfils RQ2.3). In aggregate, using Identitas and hypernormalisation should enable the easy development of large-scale ontologies in the future (this fulfils RQ3).

8.4 Final Thoughts

Ontologies seek to model parts of the world in a computationally amenable way. One recurrent issue has been understanding which part of the world we should model;

indeed, once you have understood the possibility of modelling part of the world, there is a natural desire to extend this model as widely as possible. Perhaps this is typified by systems such as Cyc [73], which has been building a comprehensive model of all parts of the world for many years.

Within biomedical ontologies, the need for this kind of modelling is also apparent. Consider the following statement, which we would like to be able to state:

> The Kent variant of COVID-19 presents an alteration of the spike protein of the virus, which makes it easier for the virus to gain entry to the cell. This results in a faster infection process, causing a more rapid onset of the disease, which in turn, increases its spread through the human population.

This is hard or impossible to model ontologically as it is so multi-scale. It describes biology at the population, organism, cell, protein and genetic levels. The OBO Foundry aims to coordinate the development of biomedical ontologies by encouraging ontologies developers adhere to shared principles. This includes using a common system of identifiers and encouraging the developers of domain-related ontologies to coordinate their efforts to produce a single artifact.

So, the OBO Foundry is attempting to resolve the problem of multi-scale ontologies, by encouraging developers to build parts of an ontology independently and orthogonally. Yet, this plan has not succeeded: despite the significant efforts of OBO Foundry, there is still considerable term overlap among the OBO Foundry ontologies [67] and, as our work has shown, there is a semantic mismatch between ontologies on closely related topics, such as chemical structures.

There are some fundamental problems here. Both the GO and ChEBI developers have been behaving sensibly; they have addressed an old question – how much detail should we model and how much should we simplify? We could model all of reality, all of time, but of course, this would mean that GO would need to model in the same way as the semantically more complex ChEBI. And no doubt, ChEBI would need to deal with a semantically more complex view that a full chemical ontology

would have. The solution to our problems cannot be to model at ever greater and greater detail; dealing with all the complexity of reality, all of the time is not an answer.

As our motivating example shows, we need to model at multi-scale. In order to achieve this multi-scale, we need to stop developing independent ontologies and bring them together. Clearly, this is difficult, but to achieve this we need not coordination but modularity. It is this, that has enabled us to build massive software systems, much larger than the biggest ontology.

Hypernormalisation and Identitas are both attempts to achieve these ends. They take ideas from software engineering but adapt them for ontologies; our hope is that they fulfil a role similar to encapsulation for software engineering. If they do, they will have a significant contribution to make in building large-scale, comprehensive and useful models of all of biology. This will, in turn, help us to harness the power of computation to enable us to understand life better in all of its complexity.

A

Classification: Additional Material

Table A.1: This table summaries the set of ontologies that are used in this thesis.

Ontology Name	Str	Description
The Gene Ontology	GO	A formal and computational representation of biological systems, including the functions of genes and genes products from different organisms.
The Chemical Entity of Biological Interest	ChEBI	The most comprehensive standardised and structured chemical terminology of biological interest.
Systematised Nomenclature of Medicine Clinical Terms	SNOMED-CT	A comprehensive and standardised health terminologies that cover most areas of medicine.
International Classification of Disease	ICD	An international classifications for health conditions including diseases, symptoms and injuries.
Rhea, the Annotated Reactions Database	Rhea	Rhea is an extensive resource of biochemical reactions in which the reactions participants are defined using ChEBI entities and their chemical structures.

B

Catalytic Activity: Additional Material

Table B.1: This table provides the set of catalytic activity (CA) classes with the equivalent classes from Rhea that we found during the investigation of the catalytic activity classes. Here we present each GO class ID with the its equivalent Rhea entry ID. There are GO classes which include more than one reactions, we have split these into classes.

No	Catalytic Activity class	Equivalents Rhea
1	GO:0002948	RHEA:54084
2	GO:0001888	RHEA:16224
3	GO:0003755	RHEA:16237
4	GO:0003810	RHEA:43771
5	GO:0030743	RHEA:43212
6	GO:0016436	RHEA:43184
7	GO:0016437	RHEA:14433
8	GO:0008825	RHEA:11991
9	GO:0047144	RHEA:14236
10	GO:1990259	RHEA:50904
11	GO:0033801	RHEA:11435
12	GO:0008882	RHEA:18592
13	GO:0033837	RHEA:35628
14	GO:0033830	RHEA:17841
15	GO:0102511	RHEA:35434
16	GO:0008412	RHEA:27782
17	GO:0030792	RHEA:11685
18	GO:0047946	RHEA:18472
19	GO:0008353	RHEA:10219
20	GO:0047961	RHEA:19872
21	GO:0050313	RHEA:12984
22	GO:0004457	RHEA:23447
23	GO:0047137	RHEA:13116
24	GO:0004455	RHEA:22071

Continued on next page...

Table B.1 – continued from previous page

No	Catalytic Activity class	Equivalents Rhea
25	GO:0008805	RHEA:13984
26	GO:0052580	RHEA:25644
27	GO:0052579	RHEA:25647
28	GO:0018678	RHEA:16388
29	GO:0008874	RHEA:23938
30	GO:0052590	RHEA:28756
31	GO:0052589	RHEA:30095
32	GO:0018665	RHEA:15160
33	GO:0003908	RHEA:24000
34	GO:0033786	RHEA:27465
35	GO:0033785	RHEA:27473
36	GO:0003968	RHEA:21251
37	GO:0036408	RHEA:21992
38	GO:0052908	RHEA:19609
39	GO:0052909	RHEA:42780
40	GO:0003976	RHEA:13584
41	GO:0050316	RHEA:13517 & RHEA:12697
42	GO:0004674	RHEA:46608 & RHEA:17989
43	GO:0003956	RHEA:19149
44	GO:0004851	RHEA:32459
45	GO:0018708	RHEA:18280
46	GO:0015667	RHEA:16857
47	GO:0033094	RHEA:12271
48	GO:0047286	RHEA:11820
49	GO:0052622	RHEA:36331
50	GO:0047284	RHEA:18361
51	GO:0047281	RHEA:10232

Continued on next page...

Table B.1 – continued from previous page

No	Catalytic Activity class	Equivalents Rhea
52	GO:0052623	RHEA:36327
53	GO:0052624	RHEA:40551
54	GO:0061599	RHEA:35047
55	GO:0047297	RHEA:19813
56	GO:0047293	RHEA:17709
57	GO:0047291	RHEA:18417
58	GO:0047228	RHEA:17285
59	GO:0004579	RHEA:22980
60	GO:0004578	RHEA:13865
61	GO:0008951	RHEA:44012
62	GO:0047237	RHEA:23465
63	GO:0008955	RHEA:23711
64	GO:0047253	RHEA:19945
65	GO:0008914	RHEA:12340 & RHEA:50416
66	GO:0051742	RHEA:38000
67	GO:0047257	RHEA:19165
68	GO:0051748	RHEA:13205
69	GO:0004577	RHEA:23380
70	GO:0047267	RHEA:28118
71	GO:0008983	RHEA:24452
72	GO:0004145	RHEA:11116
73	GO:0044605	RHEA:56080
74	GO:0043752	RHEA:15769 & RHEA:15765
75	GO:0008999	RHEA:16433
76	GO:0043754	RHEA:18865
77	GO:0043761	RHEA:35439
78	GO:0008965	RHEA:23883

Continued on next page...

Table B.1 – continued from previous page

No	Catalytic Activity class	Equivalents Rhea
79	GO:0043764	RHEA:17817
80	GO:0004125	RHEA:22728
81	GO:0008963	RHEA:21920
82	GO:0050053	RHEA:13656
83	GO:0043770	RHEA:26466
84	GO:0030409	RHEA:15097
85	GO:0043772	RHEA:34075
86	GO:0050071	RHEA:10668
87	GO:0043776	RHEA:36067
88	GO:0008976	RHEA:19573
89	GO:0043777	RHEA:34592
90	GO:0004127	RHEA:25094
91	GO:0043780	RHEA:26286
92	GO:0052654	RHEA:18321
93	GO:0052655	RHEA:24813
94	GO:0052656	RHEA:24801
95	GO:0043712	RHEA:49440
96	GO:0050004	RHEA:56344
97	GO:0052669	RHEA:51680
98	GO:0043720	RHEA:31558
99	GO:0004169	RHEA:17377 & RHEA:53396
100	GO:0008467	RHEA:15461
101	GO:0008466	RHEA:23360
102	GO:0008469	RHEA:12108
103	GO:0008476	RHEA:16804
104	GO:0008479	RHEA:16636
105	GO:0034738	RHEA:33479

Continued on next page...

Table B.1 – continued from previous page

No	Catalytic Activity class	Equivalents Rhea
106	GO:0034737	RHEA:33483
107	GO:0102425	RHEA:61212
108	GO:0047600	RHEA:34183
109	GO:0090447	RHEA:33559
110	GO:0008455	RHEA:12944
111	GO:0008454	RHEA:16060
112	GO:0008459	RHEA:11108
113	GO:0047756	RHEA:16101
114	GO:0010341	RHEA:36123
115	GO:0019161	RHEA:18220
116	GO:0102437	RHEA:25629

References

[1] B Alberts, A Johnson, J Lewis, P Walter, M Raff, and K Roberts. *Molecular Biology of the Cell 4th Edition: International Student Edition*. Routledge, 2002.

[2] B Alberts, A Johnson, J Lewis, M Raff, K Roberts, and P Walter. Carrier proteins and active membrane transport. In *Molecular Biology of the Cell. 4th edition*. Garland Science, 2002.

[3] B Alberts, A Johnson, J Lewis, M Raff, K Roberts, and P Walter. Principles of membrane transport. In *Molecular Biology of the Cell. 4th edition*. Garland Science, 2002.

[4] R Alcántara, K. B Axelsen, A Morgat, E Belda, E Coudert, A Bridge, H Cao, P De Matos, M Ennis, S Turner, *et al.* Rhea—a manually curated resource of biochemical reactions. *Nucleic acids research*, 40(D1):D754–D760, 2012.

[5] C.-A Alpert and B Chassy. Molecular cloning and dna sequence of lace, the gene encoding the lactose-specific enzyme ii of the phosphotransferase system of lactobacillus casei. evidence that a cysteine residue is essential for sugar phosphorylation. *Journal of Biological Chemistry*, 265(36):22561–22568, 1990.

[6] N Alshammry and P Lord. Identitas: A better way to be meaningless. *arXiv preprint arXiv:1709.09021*, 2017.

[7] N Alshammry and P Lord. Identitas: Semantics-free and human-readable identifiers. *Applied Ontology*, (Preprint):1–16, 2021.

[8] C. J Baker and K.-H Cheung. *Semantic web: Revolutionizing knowledge discovery in the life sciences*. Springer Science & Business Media, 2007.

[9] R Baldock and A Burger. Anatomical ontologies: names and places in biology. *Genome biology*, 6(4):1–6, 2005.

[10] J. P Balhoff, B Good, S Carbon, and C Mungall. Arachne: an owl rl reasoner applied to gene ontology causal activity models (and beyond). In *International Semantic Web Conference (P&D/Industry/BlueSky)*, 2018.

[11] A Bandrowski, R Brinkman, M Brochhausen, M. H Brush, B Bug, M. C Chibucos, K Clancy, M Courtot, D Derom, M Dumontier, *et al.* The ontology for biomedical investigations. *PloS one*, 11(4), 2016.

[12] J Bard, S. Y Rhee, and M Ashburner. An ontology for cell types. *Genome biology*, 6(2):R21, 2005.

[13] T Berners-Lee, R Fielding, and L Masinter. Uniform resource identifier (uri): Generic syntax. Technical report, 2004.

[14] T Berners-Lee, L Masinter, and M McCahill. Uniform resource locators (url). Technical report, 1994.

[15] D Binns, E Dimmer, R Huntley, D Barrell, C O'donovan, and R Apweiler. Quickgo: a web-based tool for gene ontology searching. *Bioinformatics*, 25(22):3045–3046, 2009.

[16] J. A Blake, J. T Eppig, C. J Bult, J. A Kadin, and J. E Richardson. The mouse genome database (mgd): updates and enhancements. *Nucleic acids research*, 34(suppl_1):D562–D567, 2006.

[17] E Bosdriesz, M. T Wortel, J. R Haanstra, M. J Wagner, P De La Torre Cortés, and B Teusink. Low affinity uniporter carrier proteins can increase net substrate uptake rate by reducing efflux. *Scientific reports*, 8(1):1–9, 2018.

[18] D Botstein, J. M Cherry, M Ashburner, C Ball, J Blake, H Butler, A Davis, K Dolinski, S Dwight, J Eppig, *et al.* Gene ontology: tool for the unification of biology. *Nat genet*, 25(1):25–29, 2000.

[19] K. K Breitman, M. A Casanova, and W Truszkowski. Ontology in computer science. *Semantic Web: Concepts, Technologies and Applications*, pages 17–34, 2007.

[20] L Brüggemann, P Dietrich, D Becker, I Dreyer, K Palme, and R Hedrich. Channel-mediated high-affinity k+ uptake into guard cells from arabidopsis. *Proceedings of the National Academy of Sciences*, 96(6):3298–3302, 1999.

[21] S Carbon, A Ireland, C. J Mungall, S Shu, B Marshall, S Lewis, A Hub, and W. P. W Group. Amigo: online access to ontology and annotation data. *Bioinformatics*, 25(2):288–289, 2009.

[22] J. J Carroll, I Dickinson, C Dollin, D Reynolds, A Seaborne, and K Wilkinson. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83, 2004.

[23] R Caspi, T Altman, R Billington, K Dreher, H Foerster, C. A Fulcher, T. A Holland, I. M Keseler, A Kothari, A Kubo, *et al.* The metacyc database of metabolic pathways and enzymes and the biocyc collection of pathway/genome databases. *Nucleic acids research*, 42(D1):D459–D471, 2014.

[24] B Chandrasekaran, J. R Josephson, and V. R Benjamins. What are ontologies, and why do we need them? *IEEE Intelligent Systems and their applications*, 14(1):20–26, 1999.

[25] J. H Chase, E Bolyen, J. R Rideout, and J. G Caporaso. cual-id: globally unique, correctable, and human-friendly sample identifiers for comparative omics studies. *Msystems*, 1(1):e00010–15, 2015.

[26] J. M Cherry, C Ball, S Weng, G Juvik, R Schmidt, C Adler, B Dunn, S Dwight, L Riles, R. K Mortimer, *et al.* Genetic and physical maps of saccharomyces cerevisiae. *Nature*, 387(6632 Suppl):67, 1997.

[27] G. O Consortium. Gene ontology consortium: going forward. *Nucleic acids research*, 43(D1):D1049–D1056, 2015.

[28] G. O Consortium. The gene ontology resource: 20 years and still going strong. *Nucleic acids research*, 47(D1):D330–D338, 2019.

[29] G. M Cooper and R. E Hausman. *The cell: Molecular approach*. Medicinska naklada, 2004.

[30] O Corcho and A Gómez-Pérez. A roadmap to ontology specification languages. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 80–96. Springer, 2000.

[31] M Da Silveira, J. C Dos Reis, and C Pruski. Management of dynamic biomedical terminologies: current status and future challenges. *Yearbook of Medical informatics*, 10(1):125, 2015.

[32] J Day-Richter, M. A Harris, M Haendel, G. O. O.-E. W Group, and S Lewis. Obo-edit—an ontology editor for biologists. *Bioinformatics*, 23(16):2198–2200, 2007.

[33] C Dessimoz and N Škunca. *The Gene Ontology Handbook*, volume 1446. Humana Press New York, NY, USA:, 2017.

[34] I Dreyer and E Michard. High-and low-affinity transport in plants from a thermodynamic point of view. *Frontiers in Plant Science*, 10:1797, 2020.

[35] L Du Plessis, N Škunca, and C Dessimoz. The what, where, how and why of gene ontology—a primer for bioinformaticians. *Briefings in bioinformatics*, 12(6):723–735, 2011.

[36] M Dürst and M Suignard. Internationalized resource identifiers (iris). Technical report, 2004.

[37] EBI. Urigen, 2016.

[38] S El-Sappagh, F Franda, F Ali, and K.-S Kwak. Snomed ct standard ontology based on the ontology for general medical science. *BMC medical informatics and decision making*, 18(1):76, 2018.

[39] A Fleischmann, M Darsow, K Degtyarenko, W Fleischmann, S Boyce, K. B Axelsen, A Bairoch, D Schomburg, K. F Tipton, and R Apweiler. Intenz, the integrated relational enzyme database. *Nucleic acids research*, 32(suppl_1):D434–D437, 2004.

[40] F Gandon and G Schreiber. RDF 1.1 XML Syntax. Technical report, 2014.

[41] A Gangemi and V Presutti. Ontology design patterns. In *Handbook on ontologies*, pages 221–243. Springer, 2009.

[42] A Gangemi and V Presutti. *Ontology Design Patterns*, pages 221–243. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[43] Gene Ontology Consortium. The gene ontology (go) database and informatics resource. *Nucleic acids research*, 32(suppl_1):D258–D261, 2004.

[44] A Ghazvinian, N. F Noy, and M. A Musen. How orthogonal are the obo foundry ontologies? In *Journal of biomedical semantics*, volume 2, page S2. Springer, 2011.

[45] T. R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[46] T. R Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, 43(5-6):907–928, 1995.

[47] G Grumbling and V Strelets. Flybase: anatomical data, images and queries. *Nucleic acids research*, 34(suppl_1):D484–D488, 2006.

[48] N Guarino and C Welty. Evaluating ontological decisions with ontoclean. *Communications of the ACM*, 45(2):61–65, 2002.

[49] R. P Guralnick, N Cellinese, J Deck, R. L Pyle, J Kunze, L Penev, R Walls, G Hagedorn, D Agosti, J Wieczorek, *et al.* Community next steps for making globally unique identifiers work for biocollections data. *ZooKeys*, (494):133, 2015.

[50] M Hadzic, P Wongthongtham, T Dillon, and E Chang. *Ontology-based multi-agent systems*. Springer, 2009.

[51] J Hastings, D Magka, C Batchelor, L Duan, R Stevens, M Ennis, and C Steinbeck. Structure-based classification and ontology in chemistry. *Journal of cheminformatics*, 4(1):8, 2012.

[52] J Hastings, G Owen, A Dekker, M Ennis, N Kale, V Muthukrishnan, S Turner, N Swainston, P Mendes, and C Steinbeck. Chebi in 2016: Improved services and an expanding collection of metabolites. *Nucleic acids research*, 44(D1):D1214–D1219, 2015.

[53] M. A Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.

[54] D. P Hill, N Adams, M Bada, C Batchelor, T. Z Berardini, H Dietze, H. J Drabkin, M Ennis, R. E Foulger, M. A Harris, *et al.* Dovetailing biology and chemistry: integrating the gene ontology with the chebi chemical ontology. *BMC genomics*, 14(1):513, 2013.

[55] D. P Hill, B Smith, M. S McAndrews-Hill, and J. A Blake. Gene ontology annotations: what they mean and where they come from. In *BMC bioinformatics*, volume 9, page S2. BioMed Central, 2008.

[56] P Hitzler, M Krötzsch, B Parsia, P. F Patel-Schneider, S Rudolph, *et al.* Owl 2 web ontology language primer. *W3C recommendation*, 27(1):123, 2009.

[57] M Hori, J Euzenat, and P Patel-Schneider. Owl web ontology language xml presentation syntax. 2003.

[58] M Horridge and S Bechhofer. The owl api: A java api for owl ontologies. *Semantic web*, 2(1):11–21, 2011.

[59] M Horridge and P. F Patel-Schneider. Owl 2 web ontology language manchester syntax. *W3C Working Group Note*, 2009.

[60] I Horrocks *et al.* Daml+oil: A description logic for the semantic web. *IEEE Data Eng. Bull.*, 25(1):4–9, 2002.

[61] R Iqbal, M. A. A Murad, A Mustapha, N. M Sharef, *et al.* An analysis of ontology engineering methodologies: A literature review. *Research journal of applied sciences, engineering and technology*, 6(16):2993–3000, 2013.

[62] R. C Jackson, J. P Balhoff, E Douglass, N. L Harris, C. J Mungall, and J. A Overton. Robot: A tool for automating ontology workflows. *BMC bioinformatics*, 20(1):407, 2019.

[63] M Jacobson, A. E Sedeño-Cortés, and P Pavlidis. Monitoring changes in the gene ontology and their impact on genomic data analysis. *bioRxiv*, 2018.

[64] R Jardim-Gonçalves, J Müller, K Mertins, and M Zelm. *Enterprise interoperability II: new challenges and approaches*, volume 3. Springer Science & Business Media, 2007.

[65] W Jiang and E Stefanakis. What3words geocoding extensions. *Journal of Geovisualization and Spatial Analysis*, 2(1):7, 2018.

[66] N. S Juty, N Le Novère, H Hermjakob, and C Laibe. Delivering'cool uris' that do not change. In *SWAT4LS*, 2012.

[67] M. R Kamdar, T Tudorache, and M. A Musen. A systematic analysis of term reuse and term overlap across biomedical ontologies. *Semantic web*, 8(6):853–871, 2017.

[68] M Kanehisa and S Goto. Kegg: kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 28(1):27–30, 2000.

[69] M Kanehisa, S Goto, Y Sato, M Kawashima, M Furumichi, and M Tanabe. Data, information, knowledge and principle: back to metabolism in kegg. *Nucleic acids research*, 42(D1):D199–D205, 2014.

[70] Y Kazakov, M Krötzsch, and F Simančík. The incredible elk. *Journal of automated reasoning*, 53(1):1–61, 2014.

[71] B Khoury, C Kogan, and S Daouk. *International Classification of Diseases 11th Edition (ICD-11)*, pages 1–6. Springer International Publishing, Cham, 2017.

[72] P Leach, M Mealling, and R Salz. A universally unique identifier (uuid) urn namespace. Technical report, 2005.

[73] D. B Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[74] H Lodish, A Berk, S. L Zipursky, P Matsudaira, D Baltimore, and J Darnell. Molecular cell biology 4th edition. *National Center for Biotechnology Information, Bookshelf*, 9, 2000.

[75] P Lord. Components of an ontology. *Ontogenesis*, 2010.

[76] P Lord. The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL. *OWLED 2013*, March 2013.

[77] P Lord and R Stevens. Facets, tiers and gems: Ontology patterns for hyper-normalisation. *arXiv preprint arXiv:1711.07273*, 2017.

[78] J Malone, R Stevens, S Jupp, T Hancocks, H Parkinson, and C Brooksbank. Ten simple rules for selecting a bio-ontology, 2016.

[79] D Man. Ontologies in computer science. *Didactica mathematica*, 31(1):43, 2013.

[80] N. A. A Manaf, S Bechhofer, and R Stevens. A survey of identifiers and labels in owl ontologies. In *OWLED*, volume 614. Citeseer, 2010.

[81] H Michael Damm. Totally anti-symmetric quasigroups for all orders $n \neq 2, 6$. *Discrete Mathematics*, 307(6):715–729, Mar 2007.

[82] F. P Miller, A. F Vandome, and J McBrewster. Apache maven. 2010.

[83] J Mosquera and A Sánchez-Pla. Serbgo: searching for the best go tool. *Nucleic acids research*, 36(suppl_2):W368–W371, 2008.

[84] C. J Mungall, M Bada, T. Z Berardini, J Deegan, A Ireland, M. A Harris, D. P Hill, and J Lomax. Cross-product extensions of the gene ontology. *Journal of biomedical informatics*, 44(1):80–86, 2011.

[85] C. J Mungall, C Torniai, G. V Gkoutos, S. E Lewis, and M. A Haendel. Uberon, an integrative multi-species anatomy ontology. *Genome biology*, 13(1):R5, 2012.

[86] C. K Ogden and I. A Richards. *The Meaning of Meaning: A Study of the Influence of Language upon Thought and of the Science of Symbolism*, volume 29. K. Paul, Trench, Trubner & Company, Limited, 1923.

[87] J. Z Pan. Resource description framework. In *Handbook on ontologies*, pages 71–90. Springer, 2009.

[88] B Parsia, P Patel-Schneider, and B Motik. Owl 2 web ontology language structural specification and functional-style syntax, 2012.

[89] N Paskin. Digital object identifier (doi®) system. *Encyclopedia of library and information sciences*, 3:1586–1592, 2010.

[90] A Rector. Representing specified values in owl:? value partitions? and? value sets? *W3C working group note*, 17, 2005.

[91] A Rector and L Iannone. Lexically suggest, logically define: Quality assurance of the use of qualifiers and expected results of post-coordination in snomed ct. *Journal of biomedical informatics*, 45(2):199–209, 2012.

[92] A Rector and J Rogers. Patterns, properties and minimizing commitment: Reconstruction of the galen upper ontology in owl. In *Proceedings of the EKAW*, volume 4. Citeseer, 2004.

[93] A. L Rector. Normalisation of ontology implementations: Towards modularity, re-use, and maintainability. In *EKAW Workshop on Ontologies for Multiagent Systems*, 2002.

[94] A. L Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including owl. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 121–128. ACM, 2003.

[95] A Rodriguez-Navarro, M. R Blatt, and C. L Slayman. A potassium-proton symport in neurospora crassa. *The Journal of general physiology*, 87(5):649–674, 1986.

[96] A Rodríguez-Navarro *et al.* Potassium transport in fungi and plants. *Biochimica et Biophysica Acta, Reviews on Biomembranes*, 1469(1):1–30, 2000.

[97] A Rodríguez-Navarro and F Rubio. High-affinity potassium and sodium transport systems in plants. *Journal of Experimental Botany*, 57(5):1149–1160, 2006.

[98] M. H Saier. A functional-phylogenetic classification system for transmembrane solute transporters. *Microbiology and molecular biology reviews*, 64(2):354–411, 2000.

[99] G. E Schulz. Porins: general to specific, native to engineered passive pores. *Current opinion in structural biology*, 6(4):485–490, 1996.

[100] A Sidhu, T. S Dillon, and E Chang. Protein ontology. In *Biological Database Modeling*, pages 63–80. Artech House, 2007.

[101] B Smith, M Ashburner, C Rosse, J Bard, W Bug, W Ceusters, L. J Goldberg, K Eilbeck, A Ireland, C. J Mungall, *et al.* The obo foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature biotechnology*, 25(11):1251, 2007.

[102] B Smith, W Ceusters, B Klagges, J Köhler, A Kumar, J Lomax, C Mungall, F Neuhaus, A. L Rector, and C Rosse. Relations in biomedical ontologies. *Genome biology*, 6(5):R46, 2005.

[103] K. A Spackman. An examination of owl and the requirements of a large health care terminology. In *OWLED*, 2007.

[104] D Spohr. *Towards a multifunctional lexical resource: Design and implementation of a graph-based lexicon model*, volume 141. Walter de Gruyter, 2012.

[105] S Staab and R Studer. *Handbook on ontologies*. Springer Science & Business Media, 2010.

[106] M. Q Stearns, C Price, K. A Spackman, and A. Y Wang. Snomed clinical terms: overview of the development process and project status. In *Proceedings of the AMIA Symposium*, page 662. American Medical Informatics Association, 2001.

[107] R Stevens and P Lord. Semantic publishing of knowledge about amino acids. In *Workshop on Semantic Publishing (SePublica 2012) 9 th Extended Semantic Web Conference Hersonissos, Crete, Greece, May 28, 2012*, page 45, 2012.

[108] S. H Tirmizi, S Aitken, D. A Moreira, C Mungall, J Sequeda, N. H Shah, and D. P Miranker. Mapping between the obo and owl ontology languages. *Journal of biomedical semantics*, 2(S1):S3, 2011.

[109] C Vergara, R Latorre, N. V Marrion, and J. P Adelman. Calcium-activated potassium channels. *Current opinion in neurobiology*, 8(3):321–329, 1998.

[110] D Vrandečić and Y Sure. How to design better ontology metrics. In *European Semantic Web Conference*, pages 311–325. Springer, 2007.

[111] D Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002.

[112] J. D Warrender and P Lord. The karyotype ontology: a computational representation for human cytogenetic patterns. *arXiv preprint arXiv:1305.3758*, 2013.

[113] J. D Warrender and P Lord. How, what and why to test an ontology. *arXiv preprint arXiv:1505.04112*, 2015.

[114] J. D Warrender and P Lord. Scaffolding the mitochondrial disease ontology from extant knowledge sources. *arXiv preprint arXiv:1505.04114*, 2015.

[115] J. D Warrender. *The consistent representation of scientific knowledge: investigations into the ontology of karyotypes and mitochondria*. PhD thesis, Newcastle University, 2015.

[116] S Weibel, J Kunze, C Lagoze, and M Wolf. Dublin core metadata for resource discovery. Technical report, 1998.

[117] C Welty and W Andersen. Towards ontoclean 2.0: A framework for rigidity. *Applied Ontology*, 1(1):107–116, 2005.

[118] P. L Whetzel, N. F Noy, N. H Shah, P. R Alexander, C Nyulas, T Tudorache, and M. A Musen. Bioportal: enhanced functionality via new web services from the national center for biomedical ontology to access and use ontologies in software applications. *Nucleic acids research*, 39(suppl_2):W541–W545, 2011.

[119] D. S Wilkerson. A proposal for proquints: Identifiers that are readable, spellable, and pronounceable. *CoRR*, abs/0901.4016, 2009.

[120] C. J Wroe, R Stevens, C. A Goble, and M Ashburner. A methodology to migrate the gene ontology to a description logic environment using daml+ oil. In *Biocomputing 2003*, pages 624–635. World Scientific, 2002.

[121] L Yu. *A developer's guide to the semantic Web*. Springer Science & Business Media, 2011.

[122] X. C Zhang, Y Zhao, J Heng, and D Jiang. Energy coupling mechanisms of mfs transporters. *Protein Science*, 24(10):1560–1579, 2015.