Knowledge Management Enviroments for High Throughput Biology

Abhey Shah

A Thesis submitted for the degree of MPhil

Biology Department

University of York

September 2007

Abstract

With the growing complexity and scale of data sets in computational biology and chemoinformatics, there is a need for novel knowledge processing tools and platforms. This thesis describes a newly developed knowledge processing platform that is different in its emphasis on architecture, flexibility, builtin facilities for datamining and easy cross platform usage.

There exist thousands of bioinformatics and chemoinformatics databases, that are stored in many different forms with different access methods, this is a reflection of the range of data structures that make up complex biological and chemical data. Starting from a theoretical basis, FCA (Formal Concept Analysis) an applied branch of lattice theory, is used in this thesis to develop a file system that automatically structures itself by it's contents. The procedure of extracting concepts from data sets is examined. The system also finds appropriate labels for the discovered concepts by extracting data from ontological databases. A novel method for scaling non-binary data for use with the system is developed.

Finally the future of integrative systems biology is discussed in the context of efficiently closed causal systems.

Contents

List of Tables

List of Figures

Chapter 1

Motivations and goals of the thesis

1.1 Conceptual frameworks

Biology as a subject has recently experienced a significant transformation due to the ever increasing volumes of available data produced by the automation and improvement of experimental techniques. This increase in available data neccesitates a corresponding need to be able to handle that data and transform it into workable knowledge in - ideally - an automated manner. How can knowledge be defined, represented or extracted?

Some of the difficulty within biology stems from the previous conceptual approaches that have been used to describe and define it. (per-component reductionist biology compounded by different fields within biology adopting differing terminolgy). It has in recent years with the growth of Systems Biology as a discipline it has become unclear if all phenomena within a biological system can be explained by knowing the behaviour of every constituent component. Biology has long assumed a back seat in terms of its theoretical needs compared to physical systems. However there has been an increasing awareness that biological organisms are complex systems that require different mathematical models ([99]). Mathematical modelling based on physical principles has been principally developed for (and applied to) simple systems (i.e. those that may be relatively homogenous, possess symmetry, tend to equilibirium. However biological systems have a different character - being generally heterogenous, dynamic, self-assembling and self-organizing. In turn the research upon such systems has become increasingly data driven, biological experimental techniques now regularly provide thousands of data points per an observation step. Dealing with this data has therefore become an important and integral part of biological research.

Whilst there are arguments about the subjective nature of knowledge in this thesis I adopt a Karl Popper like stance of knowledge as beliefs which have every reason to be believed true but which may be proved wrong at any time. How can knowledge in general be stored or shared? For human consumption knowledge has been stored in many diverse forms such as

words and diagrams via multiple kinds of media like oral tradition, books, websites and journals. It is more challenging however to store and retrieve knowledge in an automated manner. Since in such an automated system knowledge must be in a form suitable for non-human agents to process. Within the philosophical discipline of objectivism, a piece of knowledge can be represented by (formally defined) concepts. Where a concept is a shared classification for which at least two objects or existents share a common set of attributes. A further indepth investigation of this issue is the subject of chapter 2; where a framework - based on objectivism - is explored in depth.

The goal of extracting and representing meaning or knowledge is broader and more fundamental then scientific theory in that it can set the preconditions and structure of theory formulation. This task has been addressed by work on conceptual frameworks, which are the tools used to formulate theories and to present preferred approaches to analysis. There is typically a cyclical process of discovery where after gathering data from a set of experiments hypotheses can be generated motivating further experiments and further data gathering and analysis[29]. Such conceptual models allow higher levels of abstraction and are independent of any particular data model. At a broader level beyond scientific investigation, conceptual models and the tools that work with them explicitly aid in the following key theory processing tasks - exploring, searching, analysing, restructuring and deciding. Almost all modern science has become as a result a cross disciplinary activity, and it motivates the need for better statistical analysis, computer science algorithms and data interface methods.

At an early stage in this project the aim was to address the task of creating a generic knowledge management tool for biological purposes. This was a somewhat lofty goal within the time and resource constaints available. One of the principal datasets, to have emerged from postgenomics biology, which shares the difficulties and features of current biological datasets [1] is the measurement of gene expression. In the next section the mechanics and context of gene expression are explained.

1.2 Biological foundations

Since the sequencing of whole genomes and the landmark publishing of the human genome in 2000, we have entered the so called post-genomic era where the volume of available biological information at the molecular level is increasing at an exponential rate. This progress shows no sign of abating, in the next few years a new generation of DNA-sequencing and expression platforms is expected to become commercially available, these will allow previously unimagined feats like sequencing the entire human genome for just a thousand dollars [68]. Already there exist DNA chips that can measure the molecular abundance of whole genomes in parallel .

[1] As detailed in 1.3.2 it is multi-dimensional, complex and noisy like much other data that is available from biological investigations.

The crucial task is now to turn this information into knowledge. There is much valuable hitherto unavailable knowledge that can be uncovered from mining data across multiple experiments. The specific focus of this work is to tackle the underlying computational architecture of the tools that can be used for ascertaining structured knowledge from biological data. If experimental data is considered to be specific unknown knowledge then there is also a need to be able to express in an automated manner general known knowledge. Both these tasks need tools that allow the manipulating of units of knowledge - concepts.

1.2.1 Gene expression data

Within the nucleus of all eukaryotic organisms cells there lie genes, a current definition of a gene is a union of genomic sequences encoding a coherent set of potentially overlapping functional products[36]. The genes within the genome encode all the sequences of proteins that make up an organism. In an organism no protein is made that does not have it's code specified within the genome of the organism. Within the human genome it is believed that there are only between 25 000 and 30 000 genes, though when the human genome was first being sequenced it was initially estimated that the number of genes were in the order of 150 000.

All genes are found within the nucleus of cells arranged as long deoxyribonuclueic acid (DNA) molecules called chromosomes. The genome is then the aggregate collection of all chromosomes. There are two strands of DNA in each chromosome, wrapped around each other in a double helix. DNA consists of four chemicals, the nucleotides: adenine, thymine, guanine and cytosine, more commonly referred to by the letters A, T, G and C. Within the helix the letters form complementary base pairs. The A and T nucleotides always lie opposite each other. Likewise the C and G nucleotides are always found opposite each other.

The process of biosysnthesis of a protein from a sequence of DNA code consists of two principal steps transcription and translation. Firstly, in transcription, within the nucleus DNA is transcribed to fragments of messenger riboxynucleic acid (mRNA). These fragments of mRNA are then transported out of the nucleus. There are specific proteins called transcription factors that bind to portions of the DNA sequence and in turn initiate either increased or decreased levels of transcription. These transcription factors typically become activated under specific stimuli available from the environment, from intercellular signalling or from within the cell.

The process of translation then occurs within the cytoplasm, molecular factories called ribosomes takes these fragments of mRNA and decode them using the genetic code, from this chains of polypeptides are formed. These in turn fold to form proteins.

Within a living cell there is a dynamic flux of molecular interactions, as the cell carries out all the functions neccessary for it's survival and growth. In the midst of these reactions there are complex chains of recurring molecular activity, these are referred to as pathways. The chemicals responsible for the bulk of triggering and facilitating pathways in turn are the

proteins synthesized from the genes.

DNA microarrays and other related techniques have made it possible to uncover the expression levels of the transcribed mRNAs produced from multiple genes in parallel, thus providing insight into the molecular behaviour of a cell under multiple experimental conditions. The different conditions may be different environmental perturbations, different time points or differing sample origin. The output of these experiments has been large numbers of observed gene expression values. From this the principal desired outcome is to discover the functional roles of genes and to map the structure of the gene regulatory network.

There are however a number of reasons which are responsible for a connection between genes and their products, as outlined in [97]:

1. Presence of common cis-regulatory motifs.
2. Co-occurrence in the same metabolic pathway(s).
3. Cis-binding to common regulator(s).
4. Physical interaction.
5. Paired evolutionary conservation among many organisms.
6. Common synthetic phenotypes upon joint deletion with a third gene.
7. Sub-cellular co-location.
8. Proximity in the genome, or in bacteria and archaea, operon co-occurrence.

Gene expression data can imply relationships that have no implication for co-regulation or have a high rate of false positives. These factors neccesitate powerful and intelligent techniques to extract meaning from gene expression data. For a gene expression experiment that consists of comparing two different experimental conditions the analysis is straightforward: every significantly differentially expressed gene (whether it is upregulated or downregulated) can be inferred to be affected by the differing variables between the two experimental conditions. However when there are multiple differing experimental conditions along with multiple gene expression measurements, there is therefore a combinatorial problem of arranging or *categorising* results in a useful manner. In the second chapter a framework for dealing with this data will be described.

1.2.2 Ontology

Ontology as defined by information scientists (as opposed to philosophers) has acquired a role as a formal method for representing knowledge[41]. In particular the data structure of this kind of ontology is to represent knowledge as a set of (possibly abstract) categories and the relationship between those categories. Informally it appears there is a natural process

whereby categorising data leads to a knowledge structure. The other essential element is for some shared agreement within a community upon the knowledge categories or concepts to be used.

1.3 Knowledge based computational environments

The practical reality of the large volume of data currently being obtained in postgenomic biology is of multiple data sources in diverse formats (often optimised for one particular purpose). The result is that whilst raw data and algorithms are readily available, it may take considerable resources to utilise this data and/or connect programs to form useful workflows. In other disciplines such as computer science, engineering and statistics, users have environments like Emacs, Acme, Matlab or R respectively that have a scripting language and built in resources in order to tackle complicated problems. Typicially this includes some plugin mechanism to allow more advanced developers to develop and load in computationally intensive or more involved functionality.

This situation was identified within biological computing and motivated the BioLisp[71] system and from this it is suggested that to meet these requirements the two key elements needed are contextual data and a well defined set of idioms that can easily map a problem to a computational representation.

One way of approaching a computational platform for knowledge management can be seen in the Knowledge OS endeavour. This project's goal is to take the paradigm of an Operating System and use it to produce a platform for working with knowledge, the Knowledge OS is not therefore envisioned to be an actual OS, rather an environment which can gather together functionality to allow useful computational tasks to be undertaken. From [71]: "Whereas Unix and Windows are operating systems for ASCII strings and files, and there are operating systems for tables such as Oracle), a knowledge operating System (or "KnowOS") is an operating system for knowledge!"

The goal of both programing languages and operating systems is to abstract away the details of the services they provide to produce a consistent interface over disparate resources. The benefit is that this allows more abstract higher level constructs to be used to coordinate resources.

The central idea then of the Knowledge Operating System is that knowledge (that is interconnected, semi-structured data of all types) is the fundamental unit analgous to a file or a database table. This knowledge is persistent and thus immediately and easily available to both users and programs at all times.

This vision of a system is particularly appealing for biology and indeed the Knowledge OS's first implementation is as a biologically based toolkit - Biolisp. Biolisp has been implemented as a mixture of lisp-based scripting language, a persistent server based object store

based on a lisp machine and a web front end.[2]

In explaining their implementation choice, the developers of Biolisp argue that relational databases provide the neccessary persistence of data but are poor at dealing with changing and semi-structured domains. Likewise classical programming language can handle semi-structured data however upon execution of a program ending the program drops or releases all of the results (and intermediary products of its calculation).

The developers of Biolisp suggest that actual operating systems and the tools within operating systems are unsuitable for this use. Within this thesis the counter argument is made that this is both possible and desirable. This is motivated by the existence of two modern operating system environments, Inferno and Plan 9 that have produced elegant approaches to interacting with complex resources via persistent file-systems. Thus one of the principal aims of this thesis is to examine exactly how in particular Inferno's architecture can be adopted to provide an environment for data analysis within biology.

Within this thesis this idea is investigated by using this Inferno and developing a novel way of treating data via a more able file system; that can then provide the mechanism for storing semi-structured, changing data.

The three abstract parts to a solution of this form; interface, database and middleware will be briefly examined next.

1.3.1 Interfaces

While Graphical User Interfaces (GUI) have provided lower learning curves and ease of operation to the functionality that can be provided by a computer, a GUI's ability to scale with a large number of items and interactions can be problematic.

This tension between ease of use and scalability can be seen in one specific application: navigating or exploring a dataset. This task intuitively seems to lend itself to some form of spatially based GUI, the very terminology used to describe this task suggests a physical activity. Also it is relatively straightforward to define semantic metrics that can be used as distance metrics between visual representations of data. Based on this there have been many attempts to use visual movements analagous to spatial movements for this purpose. Ignoring applications where the search activity is itself on a spatial dataset (for example points on a map), there have been few commercial successes of large scale (where the number of data points is greater then 40) GUI based data exploration and multiple failures. [3]

The future of human computer interfaces is argued by Don Norman to be seen in the return to a (superficially) simpler approach: the search line. In direct analogy to a command line this is the interface that a search engines provide. It is possible to infer by Google's

[2]The web interface for biolisp is effectively a command line shell.

[3]Most recently on the world wide web, where for example the websites www.grokker.com (now changed to text based search), www.antartica.com (now defunct) and www.kartoo.com all offered at their inceptions spatial based data search tools. All are noteworthy for their relative failure. Grokker has now changed to a solely text based format.

consistent prescence as the most popular search (and close to becoming the most popular email) website that this simpler approach works. The method to navigate these dataspaces is by typing phrases into our browsers and invoking a search engine. However more and more commands are being made available via the search line as search engines become answer engines. It is possible for example to ask Google to define a word by prefixing 'define:' to the word. An ad-hoc search engine command language is gradually emerging. Why is this approach so much more usable? There are two reasons postulated: scalability and the inherent limitation of visual approaches compared to what is possible without them. There are however challenges that include a steep learning curve (where knowledge is on the y-axis and effort is on the x-axis.) Within this thesis the focus is exclusively on command interpreter based methods to create solutions.

1.3.2 Databases and the character of biological data

It is worth considering the nature of biological data - especially those particular attributes of biological data that preexisting tools have problems with.

Navathe et al [77] surveyed the characteristics of biological data from a computational perspective and derived a list of key features :

1. Biological data is highly complicated.
2. The amount and range of variability in data is high.
3. Schema in biological data change at a rapid pace.
4. Representations of the same data by different biologists will likely be different.
5. Most users do not require write access.
6. Most biologists are not likely to have any knowledge of the internal structure of the database or about schema design.
7. Context gives an added meaning in biological applications.
8. Users of biological information often require access to "old" values of the data.

All of the points in the list, hint that query orientated relational databases may not be the most suitable platform for storing contextual data. In addition popular relational databases place high importance on features such as data consistency and transaction integrity when handling multiple simultaneous writers. These features - neccessary for some tasks - come at a cost both in terms of performance and flexibility.

For instance point 3, which indicates a need for fluidly changing schema posits a few solutions such as either abandoning schema (not entirely a bad thing) or automatically generating schema or making schema manipulation easy and quick.

None of the issues raised in the list above demand the use of a standard normal relational database, and trying to shoehorn data into them may lead to other issues (for example the inelegance of SQL at transitive closure). Some of the features (points 3, 6 and 7) suggest an exploratory, knowledge driven system. Additionally point number 1 hints at the need for extensibility within a data base. It needs to be easy for specific implementations of algorithms such as BLAST[7] to be used within or plugged into a database.

Relational databases are inherently based on 2 dimensional arrays of data, the data in a column is homogenous and the structure of the table is consequently "flat". As developed later in the thesis in section 3.3.3 they struggle to neatly represent or query over hierarchical data. A way of addressing the need to store hierachical based data is to use the navigational or object-orientated database model. In the navigational model, records are found by following references from other records. Hence in order to specify a particular

A simple extensible system to store data can be created using flat files and multiple simple tools. This approach is often seen in UNIX system environments, where there are many common flat file based databases such as /etc/passwd or /etc/group. These take the form of files that delimit records by newlines and fields within a record by a particular character, that can then be queried and updated via tools such as awk , grep, sed and join. The data files are then equivalent to tables in relational databases.

This so called "flat file database" can be queried in a manner semantically equivalent to a select command with a where clause in SQL (Structured Query Language) by, for instance, running a tool such as grep or awk over a file. Only the fields of interest need be extracted and printed.

Comparing this to a relational database this method seems primitive: the syntax of query use are not based on natural language, data consistency and transaction guarantees don't exist and there is a bottleneck on usable size. [4]However such a technique does produce a system that is remarkably simple and easy to extend, it is fast, has equivalent expressibility of semantics as a declarative database language like SQL and with the use of more advanced file systems (such as Plan 9's fossil or OS X's time machine) can even support transaction rollback. Furthermore there is a substantial increase in the flexability and extensability of the "where clause". Which in a "flat file database" can be a sophisticated algorithm provided it can be expressed as a program, so it is possible to easily plug in the algorithms of a tool such as Blast[7].

In addition whereas relational databases are limited to a flat list of tables, a set of files also possesses directory structure so in affect a hierarchy of tables can be created. This is a neat and practical method of producing a datastore that has the graph based features of a navigational database. More relevantly for this thesis this provides a mechanism to store

[4]The bottleneck exists because I only consider linear scans of files, without indexing techniques, since that would require a different set of base tools and would require processing the files. A general heuristic attributed to M. Satyanarayanan[100], is that a key deciding factor upon whether to store data in a file system or as a database is the ratio between time spent searching and time spent using a piece of data, if the ratio is high use a database else use a file system.

categorical information.

Going further there are architectural and functional similiarities between the approach of many current popular relational databases with operating systems. Notably they both manage their own users, user privileges, processes and raw disk access. An operating system (including the combination of files, directory structure and simple tools) offers a compelling alternative to a relational or an object orientated database: it can store both highly ordered homogenous information and hierarchical data, the enviroment it provides is easier to modify and therefore it is easier to extend in order to use advanced query functionality and for the particular task of exploring data

1.4 Goals of the thesis

The aim of this thesis is to develop tools and approaches to allow concepts to be treated as the principal unit of computational analysis, and to show how such techniques can be integrated within existing information systems. In order to develop such a conceptual data store, the idea that is explored is to integrate concepts into the hiearchy of a file system.

The key technology upon which to build this system upon is the novel operating system Inferno. The key precept underlying this operating system is to represent resoures as a file system, thus this thesis tracks the motivations, the theory and the eventual development of a knowledge resource as a file system. There are specific applications for which this system is initially developed for and its usage for these is analysed - however the hope is to develop a generic toolkit. The particular application that is the focus, is its usage with biological gene expression data. A second underlying aim is to examine the theoretical foundations that underly complex systems biology, in particular the traits of self-organizing organisms.

1.5 Structure of the thesis

There are 3 important metaphysical conceptual models that underlie modern science: Objectivism, Realism and Reductionism[44]. They define how knowledge processing is carried out in the framework of scientific enquiry.

1. Objectivism - This is a philosophy created and promulgated by Ayn Rand that encompasses positions on epistemology, ethics, politics and metaphysics. Her approach to epistemology is that everything known about an object can be derived from the attributes it possesses, commonality of attributes is the basis for defining concepts. Concepts are the fundamental unit of knowledge representation rather then propositions.

2. Realism - That an upper ontology is possible of all the world and it consists solely of measurable entities. Concepts are out there to be found and named.

3. Reductionism - A doctrine that complex phenomena can be understood by understanding the properties of simpler constituent components.

For developing a toolkit for knowledge processing, these conceptual models suggest a natural way to break up the tasks that need to be carried out - three chapters in the thesis (chapters 2, 3 and 6) are each structured around one of the conceptual models and specifically on the explicit reification of the idea(s) contained within each respective model as tasks that need addressing. With objectivism the key tasks considered are:- feature extraction and concept formation. The principal tasks that are governed by realism are concept representation and identification.

Finally the results of using the platform are detailed in Chapter 5.

Chapter 2

Objectivism: Formal Concept Analysis

2.1 Epistemology of objectivism

Concepts are the crucial units of knowledge within the epistemology of Objectivism[96]. Within objectivism, a concept is just a classification for which at least two objects or existents share a common set of attributes. Once a concept has been formed it can be treated perceptually as an object complete with its own name and with implicit information about its members.

How is concept forming carried out? The method outlined in objectivist epistemology is to take the specific attributes of two or more similar members and remove the measurable aspects of any particular object to leave the particular properties that members of the same group possesses, the example given by Ayn Rand[96] is of the concept of a kitchen table. Isolate the attributes that constitute true "table-ness" -some means of support and a flat surface - and ignore the attributes like material, length or colour that are irrelevant for defining a table. Thus, forming concepts is *finding similarities* between a group of objects, defining it and *naming* it. The process of finding and forming concepts can be argued to constitute the fundamental unit of modern scientific knowledge (rather then non-contextual propositions).

This notion of a concept has in recent times been formalised by several different theoretical approaches including Formal Concept Analysis (FCA) of Rudolf Wille[115], conceptual graphs by John Sowa[106], conceptual spaces by Peter Gärdenfors[34] and even attempts to unify all these theories by Joseph Goguen[38]. In all theories, conceptual modelling is an aid to knowledge representation.

In this chapter a brief introduction to Formal Concept Analysis is presented. As part of this exposition, the basic mathematical framework for knowledge representation is introduced. A principal theorem is restated and the standard algorithm (Next Closure) used to take

a binary matrix to a hierarchical data structure is described. Whilst the algorithm is a standard and familiar algorithm in both the FCA and data-mining community there is a relative paucity of easy to understand explanations of the algorithm. Likewise the theorem's proof suffers in the translation from its original German derivation to English [33]. Furthermore this particular algorithm in its behaviour iterates through the concepts from the top of the lattice to the bottom and integrated into this is a first step towards identifying and finding putative labels for the concepts.

2.1.1 Applying objectivism

As a tool for data exploration, concepts have an intuitive representation within a computing environment as directories in a hierarchical file-system. All of these techniques and algorithms show how data can be transformed so that many of these data operations can be carried out on a hierarchical file system representation of the data. The technical details of implementation are discussed later in Chapter 4 however this chapter introduces some of the major issues in transforming a lattice-based data structure into a hierarchical form suitable for use within a file-system.

2.1.2 Gene expression data

How can this view of knowledge extraction be applied to multivariate biological data sets such as the gene expression data set introduced in section 1.2.1. Concept forming is in essence a form of clustering. Clustering methods have already emerged as one of the most popular approaches to analyse gene expression data, based on the hypothesis that similar expression profiles imply a functional relation between genes. The genes in such clusters are often assumed to be co-regulated, i.e. to share the same regulatory controls, thereby implying biological relevance. However, gene transcript levels can be correlated either by chance (due to systematic error or experimental noise) or because of indirect effects, and therefore they might not actually be directly co-regulated.

Within gene expression analysis there are multiple typical goals in which clustering-type techniques have been applied and been found useful (adapted from [67]):

1. Finding groups of genes with similar profiles across different experimental conditions.
2. Classification of a new gene within an ontological structure or a data structure that represents the network of Gene Regulation processes.
3. Grouping of conditions based on the expression of a number of genes.
4. Classification of a new sample, given the expression of the genes under that experimental condition.

The first two cases are commonly associated with functional annotation and the latter two with clinical applications. Unsupervised clustering can directly addresses goals 1 and 3. More sophisticated (and possibly supervised) approaches to clustering are necessary to tackle objectives 2 and 4, with these goals there is more of a necessity to utilise preexisting information.

2.1.2.1 Clustering techniques

Several clustering techniques have been used and developed for gene expression data sets, including self organizing maps[79], hierarchical clustering[28], fuzzy k-means clustering[35], graph theoretical methods[103] and bi-clustering[16]. All of these methods apart from bi-clustering focus on different ways of exclusively clustering data across all conditions. However the expression levels of a set of biologically related genes might only show coherency under a subset of conditions that cluster data. Due to the complexity of a biological system's interaction with its environment and the multi factorial nature of Gene regulation. Genes might not be co-regulated across all experimental conditions. Also, genes can be involved in multiple different processes, depending upon the state of the organism during a given experiment.

2.1.2.2 Bi-clustering

Bi-clustering[16] is a technique whose more recent development has been motivated by the need to carry out analysis on the co-regulation of genes, though the algorithms underlying it have been proposed and used in other fields. Bi-clustering takes both genes and experimental conditions to form *bi-clusters*. Each bi-cluster is defined by the subset of genes and subset of experimental conditions which are somehow similar. It therefore models condition-specific patterns of co-expression. In essence bi-clustering tries to retain a context to clusters of coherent gene expression values.

Retaining this context leads to identifying every bi-cluster with a concept and indeed bi-clustering in the specific case where the data is a binary context table is isomorphic to Formal Concept Analysis.

2.1.2.3 Formal Concept Analysis

Formal Concept Analysis, a form of bi-clustering, clusters both feature data and objects (into concepts) thereby modelling condition-specific attribute patterns. Further, concepts (consisting of both conditions and genes) have overlap thereby allowing the modelling of genes activity in multiple biological processes. However the context table (at least initially) for gene expression data is not binary; it is continuous. There are several different methods for transforming continuous data into a binary context table, hence a full treatment of that problem (including showing how it encapsulates other bi-clustering techniques) is delayed until

Chapter 5. There a technique that uses FCA tools in conjunction with novel transducers is used to extract features from gene expression profile data.

2.1.3 The amino acids: an example data set

To motivate and aid in the following exposition, a simple exemplar data set shown in table 2.1 that consists of all the amino acids and some of the physio-chemical attributes they possess is . Throughout the rest of this chapter this running example will be used to illustrate definitions and algorithms. This example data set is a binary table, since there are certain details with manipulating numerical gene expression data that would detract from the present discussion.

In chapter 5 some of the ways that the numerical data of a gene expression experiment can be transformed into binary form is illustrated.

Intuitively it is possible to see from table 2.1 that there is some connection and similarities between some of the different amino acids(objects) and their properties (attributes) in the table. For instance all amino acids that are *tiny* are also *small*. Some of the amino acids seem to be "closer" in similarity then others, for instance Histidine has more attributes in common with Lysine then with Phenylaline. In the following how these relationships can be derived and represented will be explored.

2.2 Basics of Formal Concept Analysis

The practical application of Formal Concept Analysis, "discovers" natural clusterings of objects by their properties. It provides a simple and straightforward way of explicitly specifying the conceptualisation of a data set as a hierarchical structure called a *concept lattice*. A concept lattice can then be used both as a navigational tool and also to discover patterns.

2.2.1 Definitions

The following theoretical exposition is based on Gaunt [33]. Unless explicitly stated otherwise all sets are assumed to be finite.

2.2.1.1 Formal context:

A *formal context* is a triple $\Bbbk := (O, A, I)$ where O and A are two sets and I is a binary relation between O and A. The elements of O are objects and each has some attributes from the set A. The attributes that each object has is defined by the binary relation I. If $o \in O$ and $a \in A$ and $(o, a) \in I$ then o has attribute a. In the rest of the following exposition O and A are assumed to be finite sets.

Table 2.3 is a representation of a formal context, where the binary relation I is the content of the binary table and O and A are the row and column labels respectively.

2.2.1.2 Partial orders:

A partially ordered set P (referred to as *a poset*) is a set S together with a reflexive, anti-symmetric, transitive binary relation on S normally labelled $\leq$: $P = \{S, \leq\}$. A trivial (though non-finite) example of a poset is the set of integers with the less then ($<$) operator. Two elements (a and $b \in A$) in a poset are *comparable* if either $a \leq b$ or $b \geq a$.

2.2.1.3 Lattices:

A *lattice* is a poset with the requirement that every two elements have a suprenum (also known as the join or the lowest upper bound, using this operator $\vee$) and infimum (also known as the meet or the greatest lower bound $\wedge$). The ordering relation relates to the meet and join via the following two equivalent definitions:

$$x \leq y \Leftrightarrow x = x \wedge y$$

$$x \leq y \Leftrightarrow y = x \vee y$$

2.2.1.4 Power sets:

A *power set* (written as $\mathcal{P}(s)$ or 2^s[1]) of a set s is the set of all subsets of a set. For example for the set $s = \{a, b, c, d\}$the power set is $\mathcal{P}(s) = \{\{\emptyset\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$. The inclusion operator ($\subseteq$) then partially orders the elements of the power set. Furthermore set intersection ($\cap$) is the meet operator($\wedge$) and set union ($\cup$) is the join operator ($\wedge$).

2.2.1.5 Formal concept:

For $U \subseteq O$, we define $U' := \{a \in A | \forall o \in U : (o, a) \in I\}$. Correspondingly for $V \subseteq A$ define $V' := \{o \in O | \forall a \in V : (o, a) \in I\}$. We can consider each prime operator as a function from the power sets of O and A.

- $f : \mathcal{P}(O) \rightarrow \mathcal{P}(A), f(U) = U' = \{a \in A | \forall o \in U : (o, a) \in I\}$
- $g : \mathcal{P}(A) \rightarrow \mathcal{P}(O), g(V) = V' = \{o \in O | \forall m \in V : (o, a) \in I\}$

For example, if $U = \{Tryptophan, Lysine\}$ then $U' = \{Hydrophobic, Polar\}$.

The compound operators $g \circ f(U)$ and $f \circ g(V)$ are now closure operators over $\mathcal{P}(O)$ and $\mathcal{P}(A)$. If we denote the compound operators by " then this means that Z$\subseteq$Z" and (Z")"=Z" for any Z in $\mathcal{P}(O)$ or Z in $\mathcal{P}(A)$. Now a *formal concept* is a pair (U, V) such that $U \subseteq O$, $V \subseteq A$, $U' = V$ and $V' = U$. The set U is then called the *extent* (it is a subset of the objects)

[1]The 2^snotation signifies the set of mappings between the set $s$and $\{0, 1\}$, each mapping represents a subset of the set s, where an element is in the subset if it maps to 1 and 0 if it is not. The set of mappings and hence the number of subsets (and equivalently the size of the power set is $2^{|s|}$.)

and the set V the *intent* (it is a subset of the descriptions) of the formal concept (U, V). When referring to a concept c let $c.extent$ refer to the set $U \subseteq O$ and $c.intent$ refer to the set $V \subseteq A$.

A concept is therefore a set of objects which all possess a set of attributes. It is not immediately obvious but every concept of a given context K has the form (U'', U') for some $U \subseteq O$ and the form (V'', V') and all such pairs form concepts. This suggests a possible method to derive all concepts from a context by examining terms (X'', X') for every $X \subseteq O$.

The set of all formal concepts is thus a family of closed subsets under the " operator and a key result within FCA is that this is a lattice. That is for any two concepts there exists a unique greatest lower bound ($\vee$) and a lowest upper bound ($\wedge$). Also that there exist a top ($\top$)and a bottom($\perp$) node, where the top concept node contains all objects and the bottom concept node contains all objects for which all attributes are true. There is a canonical set of algorithms within FCA that can be used to create a concept lattice from a given context. The problem is essentially to find all closed sets associated to a binary relation and is the same as finding all rectangular blocks in a Boolean matrix.

2.2.1.6 Hasse diagram

A *Hasse diagram* is a pictorial representation of a poset, each vertex (or node) on the diagram represents an element of the set. If x is lower in the diagram then y and if there is a direct edge joining the two vertexes this signifies that $x < y$ and there does not exist a z such that $x < z < y$.

It is also worth noting in pictorial representations, that horizontal edges are not permitted, thus given a point x in the diagram all points connected to x lying above x are by the poset ordering greater then x. A totally ordered set has a Hasse diagram consisting of just a vertical line with all elements lying upon the line.

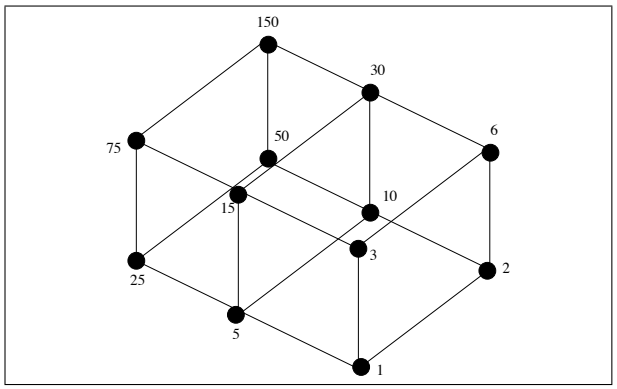


Figure 1 – *A Hasse diagram of the poset* $\{D, <\} = \{1, 2, 3, 5, 6, 15, 25, 30, 50, 75, 150\}$ *with divisibility as the ordering relation:* $x, y \in D$ *if* $x \bmod y \equiv 0$ *then* $x < y$. *All positive integers can be decomposed into lattices, the height of the lattice is the number of factors of the integer. Adapted from Leibniz's Universal Characteristic, .*

2.2.1.7 Ideals and filters

A non-empty subset I of a poset P is an *ideal*, if the following conditions hold: For every$x \in I, y \leq x \Rightarrow y \in I$ (I is a lower set). For every $x, y \in I$, there is some element $z \in I$, such that $x \leq z$ and $y \leq z$. The smallest ideal that contains a given element x is referred to as the *principle ideal* of x and is referred to as $\downarrow x = \{y \in P | y \leq x\}$. Within a Hasse diagram the principal ideal of any given element is all the elements connected to it solely by downward edges.

The dual of an ideal is a filter, again $F \subset P$ with $F \neq \emptyset$, however for every $x \in F, x \leq y \Rightarrow y \in F$, $\forall x, y \in F$ there is some element $z \in F$, such that $z \leq x$ and $z \leq y$. Also a *principle filter* of x is $\uparrow x = \{y \in P | x \leq y\}$.

2.2.1.8 Concept lattices:

Each formal context forms a conceptual hierarchy, called a *concept lattice,* the hierarchical relation between concepts is formalised by

$$(U, V) \leq (W, Z) :\Leftrightarrow U \subseteq W (\Leftrightarrow V \supseteq Z)$$

where (U, V) and (W, Z) form a concept with $U, W \subseteq O$ and $V, Z \subseteq A$.

2.2.1.9 Attribute partial order:

A useful partial order $(\leq)$ on the set A can be defined by this compatibility condition:

$$\forall o \in O, a, b \in A : (o, a) \in I, a \leq b \Rightarrow (o, b) \in I$$

This can be read as saying that if this relation holds $a \leq b$ then anything that has the attribute a also has the attribute b. This partial order is a powerful abstraction upon which to build algorithms to classify data sets and infer functional dependencies.

2.2.2 Measures

Here is a useful place to introduce some order theoretic measures, whilst they play little part in the discussions within this chapter they are intrinsic definitions that will play a part later in the thesis.

2.2.2.1 Chains and height

The following definitions are actually used in chapter 3 however they are introduced here.

A *chain* is a collection of elements all of which are comparable to each other. Recalling that two elements (a and $b \in A$) in a poset are *comparable* if either $a \leq b$ or $b \geq a$. Intuitively

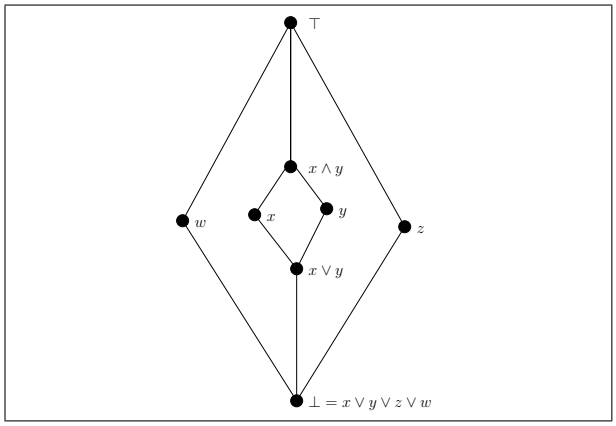


Figure 2 – *A Hasse diagram of a simple non-trivial lattice naming some of the logical relations between the nodes. Note that* $\bot < x \vee y < y < x \wedge y < \top$ *and* $\bot < x \vee y < x < x \wedge y < \top$ *however* z *and* w *are only comparable with* $\bot$ *and* $\top$.

if two nodes are comparable then there must be a monotonic vertical pathway in the Hasse diagram between them. Likewise in a Hasse diagram if it is possible to move from one node x to another y by only moving in one single vertical direction through the diagram then the two nodes are comparable.

The length of the largest chain of a poset is referred to as the *height* $\mathcal{H}$.

2.2.2.2 Anti-chains and width

Correspondingly an *anti-chain* is a collection of incomparable elements. The *width* $\mathcal{W}$ is defined as the length of the largest anti-chain. These measures are useful for giving some indication as to how efficiently a user can navigate a concept lattice or how useful the concept lattice is. For example if the lattice is very wide with a low height, this so called flat hierarchy does not offer extra navigability by its structure.

2.2.3 Simplifying attributes and feature selection

Large object attribute tables result in turn in large lattices which become unwieldy to navigate. Context tables may be simplified by combining attributes into compound attributes. This is effectively a more course-grained representation that sacrifices some details for tractability and clarity. This is done by identifying some attributes as belonging to the same group and generating a new context table. The formal definition of this action is to split up the attribute set: $A = \bigcup A_i$, $\emptyset = \bigcap A_i$, $A_i \in \mathcal{P}(A)$ equivalently $A_i \subseteq A$ and let each A_i be a set of attributes that are identified as having a similar meaning or being similar. Then define J as a binary relation between O and the $\{A_i\}$ where oJA_i if $o \in O, \exists a \in A_i$ so that oIa. Thus (O, A_i, J) is a simpler context.

Related to attribute simplification is feature selection or dimensionality reduction, a more general (in the sense that the attribute data per an object is not just binary hence - it is referred to as feature) technique of simplification. Feature selection can help to acquire better understanding about a set of data by highlighting important features and how they are related with each other. The development of algorithms for the selection of features, is an active area of research in statistics, machine learning as well as data mining[63].

2.3 Next closure algorithm implementation

In this section the next closure algorithm [33] to discover and label concepts is described, a feature of this particular algorithm is the order by which concepts are discovered. This ordering is useful in facilitating the subsequent naming of concepts.

To aid in understanding the algorithm implementation the essential steps in the algorithm are worked through with the example binary attribute data introduced in the previous sections.

The algorithm works in three stages: first the concepts are derived, secondly a lattice is formed from them and lastly the lattice is traversed to form a directed graph.

2.3.1 Calculating Closed Sets from Next Closure

Algorithm 1: $AllClosures$, calls the function $Next$ defined in Algorithm 2

Input: Binary Relation I

Output: List of Concepts

1 $concept_0 \leftarrow Next(I, \emptyset)$;
2 **repeat**
3 $\quad concept_i \leftarrow Next(I, concept_{i-1})$;
4 $\quad$ add to list
5 **until** *no more concepts to find* ;

This algorithm uses the closure operator to calculate the set of all concepts. Within 2.2.1.5 it was stated that every concept could be found by considering terms (X'', X') with $X \subseteq O$. Every concept is fully described by the attributes it possesses and every concept is a fixed point of the closure operator when applied to the attribute set. For example with $X = \{Alanine, Cysteine\}$ this gives $X' = \{hydrophobic, small\}$ and $X'' = \{Alanine, Cysteine, Threonine, Valine\}$. Note that $X''' = X' = \{hydrophobic, small\}$. An obvious strategy would be to simply calculate (X'', X') for every $X \subseteq O$.

Whilst the next function could just iterate over all the subsets a more efficient method is possible based on *lectically ordering* the subsets of the Objects O. To order the subsets, first since O is a finite set of size n, it is possible to represent each element of O as unique integer, so $O = \{1, 2, ..n\}$. From now on the notation is abused by treating any element $i \in O$ as the unique integer that the element i maps to in the set $\{1, 2, ..n\}$.

Table 1 – *The 20 naturally occurring essential Amino Acids in rows and some physio-chemical attributes (columns).*

Amino Acid	Hydrophobic	Positive	Negative	Polar	Charged	Small	Tiny	Aromatic	Aliphatic
Alanine	x					x	x		
Cysteine	x					x			
Aspartic acid			x	x	x	x			
Glutamic acid			x	x	x				
Phenylalanine	x							x	
Glycine	x					x			
Histidine	x	x		x	x			x	
Lysine	x	x		x	x				
Isoleucine	x								x
Leucine	x								x
Methionine	x								
Asparagine				x		x			
Proline						x			
Glutamine				x					
Arginine		x		x	x				
Serine				x		x	x		
Threonine	x			x		x			
Valine	x					x			x
Tryptophan	x			x				x	
Tyrosine	x			x				x	

This defines a total ordering on the set O (though the mapping can be quite arbitrary) . A subset $A \subseteq O$ is lectically smaller then a subset $B \neq A$ if the smallest element which distinguishes A and B is in B. Formally

$$A < B :\Leftrightarrow \exists_{i \in B \setminus A} A \cap \{1, 2, ..., i-1\} = B \cap \{1, 2, ..., i-1\}$$

Returning to the amino acid example let the mapping from Objects (amino acids) to integers be based on the row number from the table, so $Alaline = 1$, $Cysteine = 2$, etc. Then if $A = \{Alanine, Cysteine, Valine\}$ and $B = \{Alanine, Cysteine, Threonine\}$, this is equivalent to $A = \{1, 2, 18\}$, $B = \{1, 2, 17\}$ so that $A < B$. This can be seen in Figure 2.7, where the sets A and B are represented as bit vectors.

Index	Object	A	B
0	Alanine		
1	Cysteine	x	x
2	Aspartic Acid	x	x
3	Glutamic Acid		
4	Phenylalanine		
5	Glycine		
6	Histidine		
7	Lysine		
8	Isoleucine		
9	Leucine		
10	Methionine		
11	Asparagine		
12	Proline		
13	Glutamine		
14	Arginine		
15	Serine		
16	Threonine		
17	Valine		x
18	Tryptophan	x	
19	Tyrosine		

Table 2 – *A and B as bit vectors. $A < B$ since the smallest element that distinguishes them lies within B.*

Next define

$$A <_i B :\Leftrightarrow (i \in B, i \notin A, \forall_{j<i}(j \in A \Leftrightarrow j \in B))$$

It is easy to see that $A < B \Leftrightarrow \exists i \in O : A <_i B$. Less obvious (and because of the total ordering nature of the $<_x$ operator) , if $i < j$, $A <_i B$ and $A <_j C$ then $C <_i B$.

Final Definition:

$$A \oplus i := ((A \cap \{1, 2, ..., i-1\}) \cup \{i\})''.$$

The $\oplus$ operator ignores all of the objects greater then i, ensures i is in the set then calculates the closure of the set.

Theorem: Next Closure The smallest extent larger (with respect to lectical ordering) then a given set $A \subseteq O$ is $A \oplus i$ where i is the largest element of O with $A <_i A \oplus i$.

Proof: This can be proved by considering the following three propositions:

1. $i \notin A \Rightarrow A < A \oplus i$
2. $A <_i B$ and B is an extent $\Rightarrow A \oplus i \subseteq B \Rightarrow A \oplus i \leq B$
3. $A <_i B$ and B is an extent $\Rightarrow A <_i A \oplus i$

Let A_{+1} be the smallest extent after A with respect to the lectic order. Since $A < A_{+1}$ there exists an i such that $A <_i A_{+1}$. Using this i and from (3) $A <_i A \oplus i$ and this in turn implies by (2) that $A \oplus i \subseteq A_{+1}$ and $A \oplus i \leq A_{+1}$. However $A \oplus i$ is an extent so $A_{+1} \leq A \oplus i$ since A_{+1} is the smallest extent thus $A \oplus i = A_{+1}$.

Finally by contradiction it is straightforward to prove i is the largest element. Let $A \oplus i$ be the smallest extent larger then A and let $i < j$ so that i is no longer the largest element then since $A <_i A \oplus i$ and $A <_j A \oplus j$ then $A \oplus j <_i A \oplus i$. Thus contradicting $A \oplus i$ being the smallest extent.

Whilst a concept can be described using just the attributes it possesses (the extent) as part of the calculation the objects that are a part of the concept (the intent) are calculated too.

Algorithm 2: $Next$ called by $AllClosures$

Input: Binary Relation, oldconcept
Output: New concept or terminate

```
1 while concept not found and o ∈ Objects do
2     if o ∉ oldconcept.objects then
3         put o in oldconcept.objects ;
4         concept ← oldconcept.objects'';
5         If j ∉ concept \ oldconcept.objects : j < i then output concept ;
6     else
7         remove o from oldconcept.objects;
8     end
9 end
```

Intuitively the algorithm starts at the top of the concept lattice adding attributes to candidate subconcepts, testing them via the closure operator and adding concepts this can be seen by considering the algorithms behaviour on the example data set.

The bottom most concept node ($\top$) is defined to always exist and is defined to be the closure of all objects, that is the concept formed by (O'', O'). So in the example data set

$O = \{Alanine, Cystine, ..., Tyrosine\}$

$O' = \{\emptyset\}$ and

$O'' = O$.

In this case no attribute is possessed by all the objects, the first concept then has the empty set for the attributes, all objects match this null description.

The next iteration of the algorithm adds one object element at a time starting with the lectically highest. Once a concept is found then, then the closure of the concept with an extra attribute is then tested to see if it is a concept. Then the process of adding attributes, calculating the closure and testing is repeated until all the concepts have been uncovered. This can be best seen in table 3, which charts the progress of the algorithm in finding concepts.

Each line of the table shows the subset of O being tested for closure, each line is formed with $A \oplus i$ where A is the extent of the last found concept. If $A < A \oplus i$ then the extent is added to the concept lattice. The first 7 concepts are shown on both the table and Figure 2.3.1. The algorithm proceeds upwards through the candidate concepts levels, for each concept it can call a findterms function described in the next chapter to assign candiate terms to the concept.

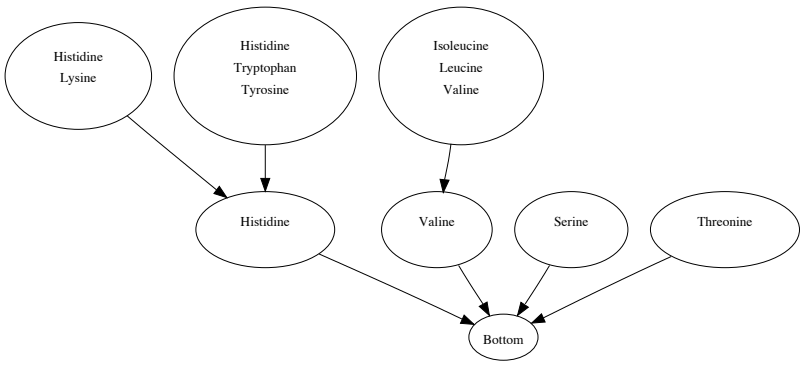


Figure 3 – *The first seven (non-trivial) concepts found by the Next Closure algorithm, they are all at the very bottom of the concept lattice.*

2.3.1.1 Algorithmic Complexity

The algorithm is dominated by the cost of calculating closures. Finding the maximum sized concept (and in turn generating the whole concept lattice) is equivalent to finding the maximum edge biclique in a bipartite graph, a problem known to be NP complete. In turn the

complexity of this algorithm is $o(n^2)$.

Stage	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	concept
A_0																					$= \emptyset$
$A_0 \oplus 19$							x												x	x	
$A_0 \oplus 18$							x												x	x	
$A_0 \oplus 17$																		x			$= A_1$
$A_1 \oplus 19$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_1 \oplus 18$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_1 \oplus 16$																	x				$= A_2$
$A_2 \oplus 19$							x	x									x		x	x	
$A_2 \oplus 18$							x	x									x		x	x	
$A_2 \oplus 17$	x	x				x											x	x			
$A_2 \oplus 15$																x					$= A_3$
$A_3 \oplus 19$			x	x			x	x				x		x	x	x	x		x	x	
$A_3 \oplus 18$			x	x			x	x				x		x	x	x	x		x	x	
$A_3 \oplus 17$	x	x	x			x						x	x			x	x	x			
$A_3 \oplus 16$			x									x				x	x				
$A_3 \oplus 14$							x	x							x						
$A_3 \oplus 13$			x	x			x	x				x		x	x	x	x		x	x	
$A_3 \oplus 12$	x	x	x			x						x	x			x	x	x			
$A_3 \oplus 11$			x									x				x	x				
$A_3 \oplus 10$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_3 \oplus 9$									x	x								x			
$A_3 \oplus 8$									x	x								x			$= A_4$
$A_4 \oplus 19$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_4 \oplus 18$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_4 \oplus 16$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_4 \oplus 15$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_4 \oplus 14$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_4 \oplus 13$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_4 \oplus 12$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_4 \oplus 11$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_4 \oplus 10$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_4 \oplus 7$							x	x													
$A_4 \oplus 6$							x														$= A_5$
$A_5 \oplus 19$							x												x	x	
$A_5 \oplus 18$							x												x	x	$= A_6$
$A_6 \oplus 17$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_6 \oplus 16$							x	x									x		x	x	
$A_6 \oplus 15$			x	x			x	x				x		x	x	x	x		x	x	
$A_6 \oplus 14$							x	x							x						
$A_6 \oplus 13$			x	x			x	x				x		x	x	x	x		x	x	
$A_6 \oplus 12$			x	x			x	x				x		x	x	x	x		x	x	
$A_6 \oplus 11$			x	x			x	x				x		x	x	x	x		x	x	
$A_6 \oplus 10$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_6 \oplus 9$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_6 \oplus 8$	x	x			x	x	x	x	x	x	x						x	x	x	x	
$A_6 \oplus 7$							x	x													$= A_7$

Table 3 – *Next Closure Algorithms progression for the first seven (non-trivial) concepts. Each of the numbered columns represents a different amino acid. Each row $A_i \oplus j$ represents a candidate extent where each A_i is the extent of a concept. If $A_i < A_i \oplus j$ then $A_{i+1} = A_i \oplus j$ and A_{i+1} is a new superconcept whose direct subconcept is A_i.*

2.3.2 Constructing the lattice

Each concept is described by its extent and its intent, then by traversing over the concepts it is straightforward to check which concept is a *subconcept* or *superconcept* and from this assemble the lattice C. The convention is for a subconcept to contain fewer objects and more attributes then its superconcept. If $c, d \in C$ then if $c.extent \subseteq d.extent$ [2]then c is a subconcept of d and d is a superconcept of c. While if $c.intent \supseteq d.intent$ then again c is a subconcept of d and d is a superconcept of c.

Within the lattice there are always two added concepts the top and the bottom: the top is defined to have all objects (and hence may have no attributes that are true for all objects); the bottom is objects for which all attributes are true.

[2]Recall that $c.extent \subseteq O$ so the extent refers to the objects within a concept. While $c.intent \subseteq A$ and the intent refers to the attributes.

By starting at the top concept (the one that is defined to have all objects) and checking all concepts below it, for concepts that are subsets of it without being subsets of a concept in between (i.e for $x, y, z \in C$ if $x \subseteq y$and there is no z s.t. $z \subseteq y$ and $x \subseteq z$). These concepts are immediate neighbours in the concept lattice and are defined to have an edge between them. The concept lattice algorithm moves through the lattice by working from the top down, this is useful for the purposes of assigning a label to the concepts, since this is in effect an initial sweep through all the concepts in the lattice.

The concept lattice may be represented by the *Hasse diagram* (see 2.2.1.6) as in figure 2.3 a) and b).

2.3.3 Unfolding the lattice to form a hierarchy

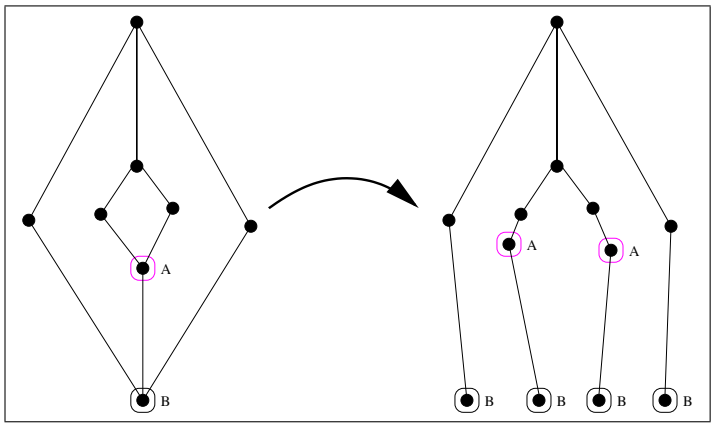
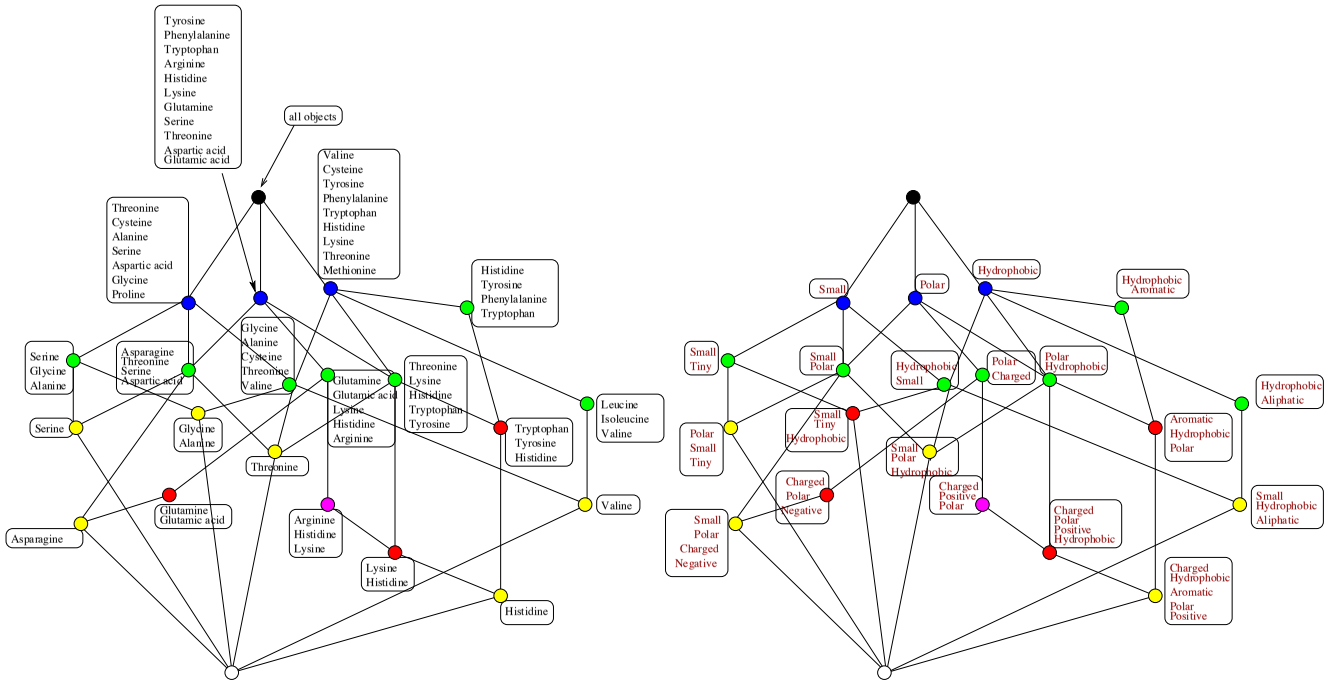


Figure 5 – *Nodes A and B are transformed by unfolding the lattice to multiple nodes.*

The concepts form a *lattice* which means that concepts can and do have multiple parents. In order to use concepts as directories in a file system (which can only have a single parent) it is necessary to transform the lattice into a rooted tree which has multiple homomorphic edges. The technical details of how it is still possible to access the lattice structure under regular usage is explained in 4.4.4. In order to initialize this poly-hierarchical file system structure every path through the lattice is traversed. A depth first traversal is made over the lattice and for each path a directory is created. This means the order of names that make up a path from the root of the mounted file system is irrelevant and there are multiple equivalent directories.

From the edges defined between the concepts, to derive a hierarchical representation of the concepts. A recursive procedure iterates through the concepts and adds all the subconcepts. A check is made to ensure that concepts which have multiple paths starting from the root only have one path added to the hierarchy. The result is a representation as shown in figure 2.5, where every edge joining two nodes represents a different possible path between the two nodes. The number attached to each edge is the order in which the edge is traversed.



(a) *lattice with all extents* **(b)** *lattice with all intents*

Figure 4 – *Concept Lattice representing the basic 20 Amino Acids with objects marked in subfigure a and attributes in subfigure b.*

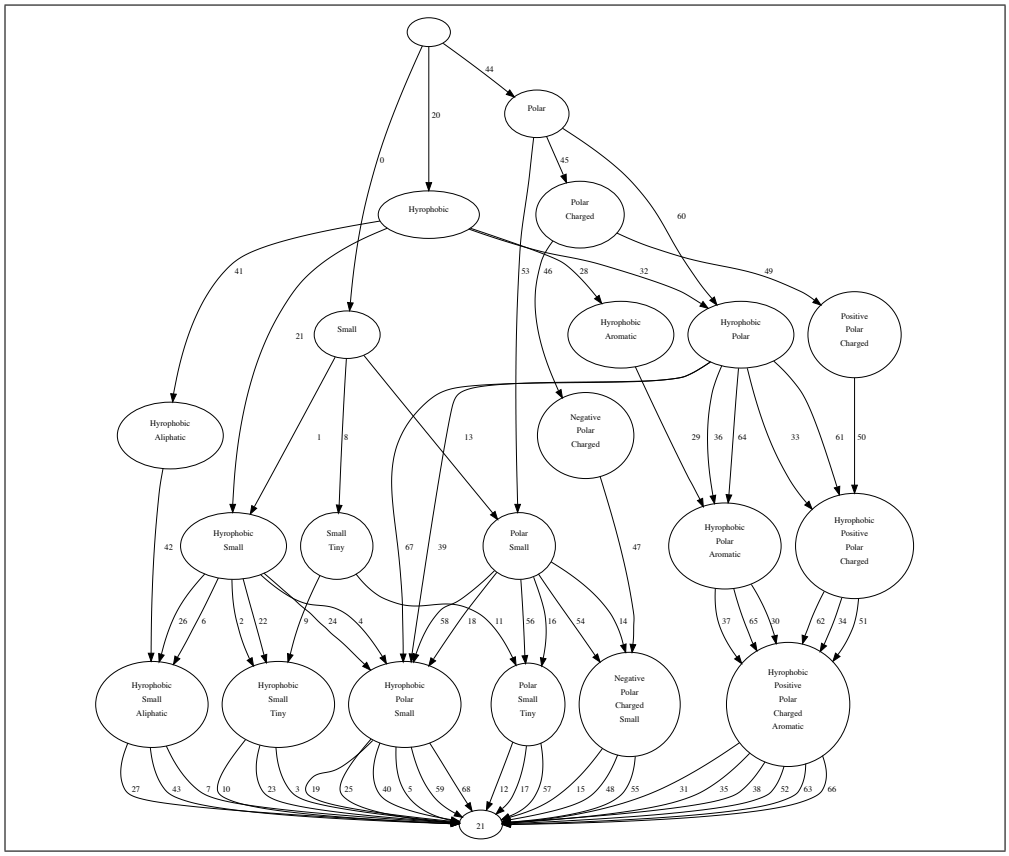


Figure 6 – *The Amino Acid lattice is transformed into a hierarchy, every unique path from one node to another is enumerated.*

2.4 Labelling concepts

Now it is possible to see how the attributes of a set of objects can induce the construction of a hierarchical data structure, however some issues remain. Each concept maps to multiple directories in the unfolded hierarchy and each concept is defined by multiple attributes. Therefore this leads to the question of which attribute should be used to name each directory.

2.4.1 Reduced labelling for concept lattices

Returning to concept lattices, there is a great deal of redundant data within the labelling of the nodes. With the following two observations:

1. If a node $c \in C$ is labelled with some attribute a, then $a \in \{e | e \in C, c \leq e\} \equiv\downarrow c$. Any attribute associated with a concept node also applies to the connected nodes below it. The concepts with this attribute form a (principle) ideal of the poset.

2. If a node $c \in C$ has $o \in c.extent$, then $o \in \{d | d \in C, d \leq c\}$. That is an object associated with one concept node within the lattice is associated with all nodes that are above it. Concepts that contain an object form a (principal) filter2.2.1.7 of the poset.

A convention can be adopted of only placing an object name upon a concept lattice once at the principal element of the filter with no loss of information. Equivalently an attribute label only need to be placed at the principal element of the ideal associated with that attribute. This is referred to within the literature as the *Galois Sub-Hierarchy [33].*

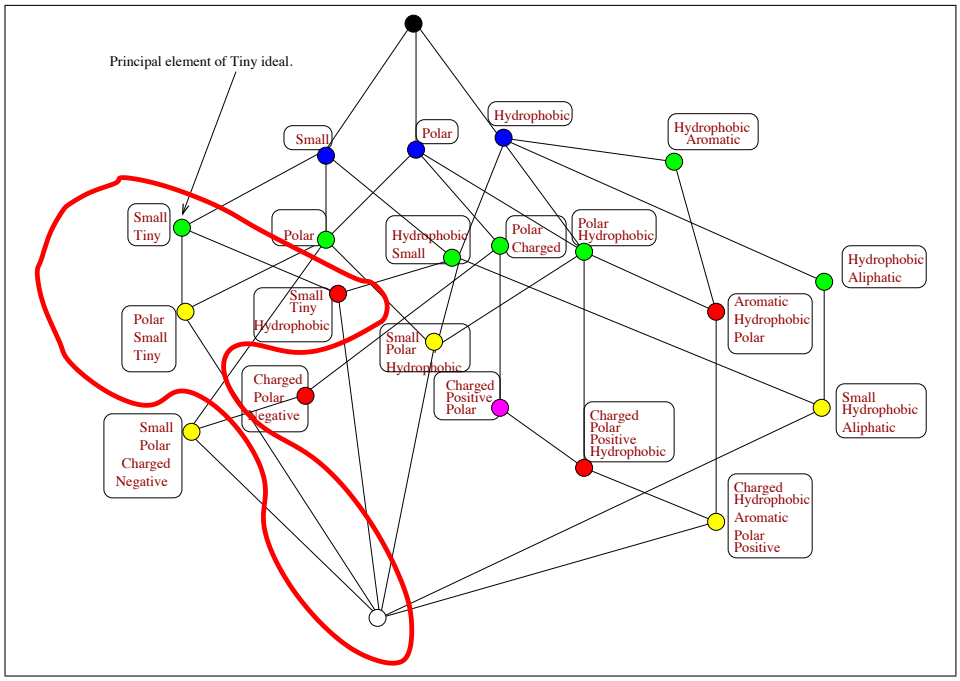


Figure 7 – *The principal ideal of* $Tiny$*. Note that nodes that are below* $tiny$ *and connected are in the principal ideal. These nodes are surrounded by the red border in the diagram.*

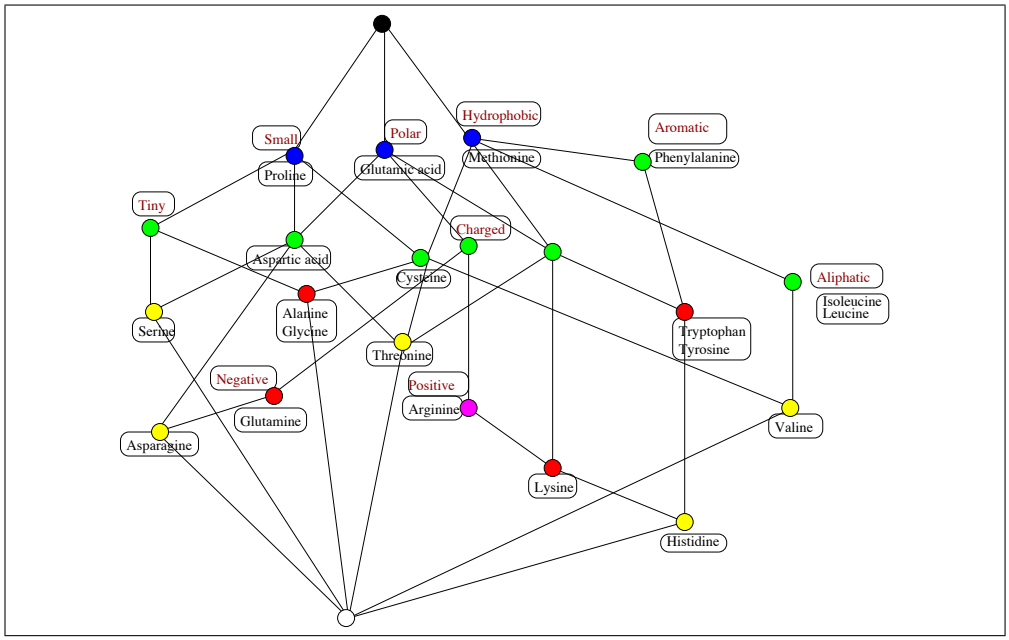


Figure 8 – *Reduced labelling of a Concept Lattice. Each attribute tag in red is inherited down the lattice to all nodes in the principal ideal of the labelled tag. Each object tag in black is inherited up the lattice to all nodes in the principle filter of the labelled tag.*

2.4.2 Naming

Whilst reduced labelling makes a more tractable readable concept lattice, in the context of a file-system this is the solution to a global problem whereas what is required with the unfolded hierarchy of a file-system is a local solution. Since each concept has multiple directories associated with it, it is possible to give a concept as many names as there are edges into the concept node. However each name (and indeed the path to a file) has to reflect the attributes possessed by objects within the directory.

2.5 Non-binary data

Formal Concept Analysis has been successfully used in many applications partly because only relatively simple mathematical concepts are needed — all that is needed to represent a data set as a concept lattice is the existence of a binary relation between two sets. However as I stated earlier biological applications have to work with complex data-sets. Data is not always in a binary form, and so various solutions have emerged from the FCA community to allow FCA to handle a non binary relationship between two sets. To model this situation, define a multi-valued context $\mathbb{K} := (O, A, V, I)$ with O, A and V as sets of Objects, Attributes and Values respectively. The relation $I \subseteq O \times A \times V$ is such that if $(o, a, v_1) \in I$ and

$(o, a, v_2) \in I$ then $v_1 \equiv v_2$.

One way to use multi-valued (any non-binary) data with FCA is to "scale" the data that is to transform each possible value into a binary form [15].

This transformation is an act of interpretation and there are several techniques to doing this. One way is to use a *conceptual scale*. A conceptual scale provides a set of binary attributes for any real valued attribute, in essence it is a mapping from the multi valued attributes V to A_{Scale}. Where A_{Scale} is the attributes of the conceptual scale. It can be easily represented by a formal context:$(O_{Scale}, A_{Scale}, I_{Scale})$ with the objects $O_{Scale} \in V$ as the possible values in the multivalued context. Then $I_{Scale} \subseteq O_{Scale} \times A_{Scale}$ is the mapping for each of the values of the multivalued attribute.

2.5.1 Nominal scaling

	a	b	c	d	e
condition 1	1	1	0	4	4
condition 2	1	1	0	4	4
condition 3	0	2	2	2	0
condition 4	5	5	0	0	0

Table 4 – *A numerical context table, representing a set of constant valued blocks.*

Here every attribute has a different range of discrete values. For each attribute a different nominal scale is used. Each multivalued attribute generates a set of binary attributes.

	a_0	a_1	a_5
0	x		
1		x	
5			x

(a) *Scale for a*

	b_1	b_2	b_5
1	x		
2		x	
5			x

(b) *Scale for b*

	c_0	c_2
0	x	
2		x

(c) *Scale for c*

	d_0	d_2	d_4
0	x		
2		x	
4			x

(d) *Scale for d*

	e_0	e_4
0	x	
4		x

(e) *Scale for e*

Table 5 – *Nominal Scales for attributes*

Nominal scales for all the other attributes produces the binary attribute table in table 6.

condition	a_0	a_1	a_5	b_1	b_2	b_5	c_0	c_2	d_0	d_2	d_4	e_0	e_4
1		x		x			x				x		x
2		x		x			x				x		x
3	x				x			x		x		x	
4			x			x	x		x			x	

Table 6 – *A nominally scaled attribute table*

2.5.2 Ordinal scaling

If an attribute has a simple linear ordering between values an *ordinal scale* can be adopted. It is easiest to describe by example. Given a simple multivalued attribute as in Table 2.5.2

	Size
Note	very small
Article	small
Thesis	large
Encyclopedia	very large

Table 7 – *A singe multivalued context of bodies of text.*

	very small	small	large	very large
Note	x	x		
Article		x		
Thesis			x	
Encyclopedia			x	x

Table 8 – *An ordinal scaling.*

Ordinal scaling can be seen implicitly within a subset of the original running example data-set, everything tiny is also small.

2.5.3 Logical scaling

If the data is in the form of Table 2.5.3 where there are actually overlapping blocks or sub matrices of similar valued attributes. Where the blocks overlap the result is the addition of the value of the block.

	a	b	c	d	e
condition 1	1	1	0	4	4
condition 2	1	3	2	6	4
condition 3	0	2	2	2	0
condition 4	5	5	0	0	0

Table 9 – *A numerical context table, representing a set of overlapping constant valued blocks.*

This can be transformed into Table 2.9, in this form the Next Closure Algorithm can easily detect the overlapping blocks.

	a_1	a_5	b_1	b_2	b_5	c_2	d_2	d_4	e_4
condition 1	x		x					x	x
condition 2	x		x	x		x	x	x	x
condition 3				x		x	x		
condition 4		x			x				

Table 10 – *Transformed attribute table*

2.6 Discussion

There are many other methods of forming clusters, especially when there is uncertainty about the attributes an object possesses. However the Next Closure algorithm is a good base upon which to represent many data analysis approaches. It is the same algorithm that is used in frequent item set data mining (with different terminology). Whilst there are several different algorithms that given a binary relation matrix (or equivalently the closure operator) uncover all concepts and generate a lattice in this chapter the principle focus has been on describing and explaining the Next Closure algorithm, Charm-L [119] for example assembles the lattice as it finds concepts. The choice to describe in detail the Next Closure algorithm was motivated in part to aid in understanding how concepts are formed and also since in a later chapter 4.4.4 a modification is made to it to facilitate the naming of concepts. This is also why the two steps of discovering concepts and assembling a lattice are kept separate, since later in the actual implementation of the algorithm the separation facilitates the operation of assigning labels to the concepts.

Already three of the major issues regarding navigating and traversing a concept lattice as a file system hierarchy have appeared:

1. The existence of multiple paths to every file object and therefore the necessity to develop mechanisms and methodologies to handle this.

2. The requirement for strategies to preprocess data into binary form suitable for applying the Formal Concept Methodology.

3. The need for heuristics or additional context in order to usefully label concepts.

The representation of concepts as a file system is explored and discussed in chapter 4. The necessary scaling and reprocessing in order to apply the formal concept methodology is demonstrated in chapter 5. Tackling the question of labelling is the subject of the next chapter.

Chapter 3

Ontology: algorithms, metrics and representations

The last chapter described an approach to extracting knowledge from data by conceptual clustering. There was a need identified to be able to find suitable labels for concepts (and in turn the directories they represent). To recap, each separate directory (representing a bi-cluster) of the concept file system can in order to facilitate navigation have a name or a label that best describes the set of experiments contained within it. The common factor in these experiments are the set of genes within those experiments that have some common behaviour. While it is possible to name a directory just by concatenating the names of the genes or of the experimental conditions, in practise this may not necessarily provide insight into the commonality underlying the genes and experimental conditions.

The key general question then is how can a useful name be found for a newly derived formal concept. The method adopted here relies upon linking the newly discovered concept with names from some vocabulary of similar preexisting concepts. This obviously relies upon some link between the attributes (or objects) in the concept and the reference vocabulary. Considering this question or the specific data-set of gene expression experiments exactly such a vocabulary exists in the form of the Gene Ontology. Furthermore the Gene Ontology has it's own structure which can in turn help provide good candidates for the names of concepts. How these labels are chosen (or created) for concepts is the main focus of this chapter.

A brief overview is given of the study of ontology and how it can be brought to bear upon this problem. After introducing Ontologies, the representation of an ontology via mathematical tools such as ordered sets and graph theory is explored. Some of the challenges inherent in computational interactions with these kinds of data sets is considered. The criterion under which the Gene Ontology can be used is discussed and a set of heuristics for using the Gene Ontology is proposed. Finally the changing usage of Ontologies, and some of the future developments of Ontologies and their related tools and database is examined. The principal

contribution of this chapter is then a technique for assigning labels to concepts.

3.1 Introduction to ontology and applications

3.1.1 Ontology

Ontology as a philosophical discipline was originally concerned with the study of being and existence[60]. In more recent times with the advent of greater computational power and the growth of knowledge resources it has also acquired a usage as a formal method for representing knowledge[41]. Ontology as a philosophical ideal can only be asymptotically approached by human knowledge and this has led to some divergence between the definition of the term in philosophy and in computer science (see [84] for further details). In terms of the activity of both groups, philosophers have primarily debated whether it is possible to assemble an ontology, whereas computer scientists have actually tried to build ontological-like structures (for example the Standard Upper Merged Ontology [80]). The question of building Ontologies is returned to towards the end of this chapter, however for the remainder of this chapter the term ontology is used in the formal (i.e. the computer science) sense of the word[41]: [an] "explicit formal specification of a shared conceptualization."[1]

A major argument against the use of ontologies is the distance between the nature of their subject compared to the representation provided by the ontology. This is similar to the arguments against the abilities of artificial intelligence. Ontologies (and related formal mechanisms such as the semantic web) are a way to describe things in a way that a computer can process; this processing occurs via application of syllogistic logic. The argument runs that there are inherent limits to the ability of such techniques, in particular the use of syllogistic reasoning. Is it possible for the semantics of something to be fully captured within syntactical statements? In general the answer is unclear and for the practical purposes of this thesis an ontology is mostly regarded as "a data model that represents a set of (possibly abstract) categories and the relationship between those categories."[41].

Ontologies complement the knowledge gleaned from a particular dataset (which will be specific and focused) with more general knowledge. This underlines the ability of ontologies to assist in sharing data, uniting disparate information systems together and providing background context.

Within this thesis this last point can be seen by the following loose argument: since the categories in an ontology are formed from the invariance of attributes of the elements, within them, and if the only relationship between terms is based on a set inclusion relation, then the representation of ontology is equivalent to the concept hierarchy introduced in the previous chapter (Chapter 2). Indeed within knowledge representation literature the categories of an ontology are named concepts. However whilst the concept hiearchy is specific to a particular

[1] Ontology, despite being formally specified, is frequently a transient entity reflecting the information that is *currently* known.

dataset the ontology is a universal entity and can provide a frame of reference upon which to "drape" or "stretch" the concept hierarchy. The benefits are that any specific knowledge applicable to a concept can be matched to the global information available from an ontology.

3.1.2 Structure and implementation of open biomedical Ontologies

The Open Biomedical Ontologies (OBO) consortium is an umbrella organization for ontologies and structured vocabularies in biology and the biomedical domain. [2]

Ontologies have taken a more specific description within knowledge management (and its application within biology) as a taxonomically organized (or structured) vocabulary of terms upon which some automated reasoning and machine based inference can be carried out. Ontologies, in this sense, have three major uses [19] : communication, automated reasoning and representation and reuse of knowledge. Ontologies in this pragmatic sense, can be seen to be specially structured databases. Many researchers ignore the structure of an ontology database and simply treat it as a list of categories[120]. However a growing body of research is directly focused on the use of the structure of ontologies.[58, 50]

The two key elements in many ontologies and specifically in all ontologies defined within the OBO are a set of *controlled terms* and a set of *relations* between the terms. Thus all ontologies that make up the OBO have a graph (G) structure where the terms are nodes (V) in the graph, and the relationships between them are arcs (E) therefore a graph $G =< V, E >$ is used in this work[58]. The nature of the relations between terms in different ontologies has been analysed and indeed an ontology constructed of the relations [105].

3.1.3 Ontologies for labelling

An ontology is a useful data structure since it provides a source of preexisting contextual background knowledge. Specifically in this project the ontology is a source of labels for clusters of items derived from a data-set. What is needed in order to use an ontology in this way? First some way of mapping from items in the data-set to the ontology, secondly some method of choosing an appropriate label given a possibly large set of terms from an ontology.

3.1.4 The Gene Ontology

The Gene Ontology[8] (GO) is one project within the Open Biomedical Ontologies (OBO). The GO is a collaborative effort to address the need for consistent descriptions of gene products in different databases and to facilitate interoperability and linkage of databases[8]. Whilst the GO consortium originally started as a collaboration between three model organisms Fly-Base(Drosophilia) the Saccaromyces Genome Database and the Mouse Genome

[2] OBO is primarily concerned with ensuring consistency and orthogonality between the ontologies. Further details about it can be found here http://www.obofoundry.org/.

Database, the curators of the content of this ontology have always strived to remain as species-independent as possible[9].

3.1.4.1 Structure

The GO project consists of three ontologies that describe gene products in terms of:

- *Biological Processes*, (BP) e.g. mitochondrial genome maintenance[3]
- *Cellular Components* (CC) e.g. the repairosome[4] and
- *Molecular Functions* (MF) e.g. metal ion transporter activity [5]

The GO currently (as at 2007) only uses two relations: 'is a' and 'has part'. There is only one link between the different ontologies: each of the three ontologies of the Gene Ontology shares a common root. The Gene Ontology by design excludes protein related domains, histological features larger then cellular components and evolutionary categories[noble] .

The Molecular Function and Biological Process branches refer to ongoing processes and the "has part" relation specifically refers to actual physical objects that have have some discrete portion, thus the 'has part' is never used within these two branches. The Cellular Component branch uses the 'has part' relation whilst the other two ontologies use the 'is a' relation. Both these relations are anti-symmetric . This implies that a relation between two terms has a direction. This restricts the graph representation of the GO to being some form of directed graph. Both relations are also transitive. Furthermore there are no cyclical relations within the ontology thus the most general combinatorial object that can represent them is a partially ordered set $P = (S, \leq)$ [50]. Recalling from section 2.2.2.1, where posets were introduced as a set S together with a partial ordering $\leq$ between elements of S, then the elements of the set S are the terms of the ontology and the relation defines the $\leq$ operator. Where if $x, y \in S$ are linked by the 'is a' or the 'has part' ontology relation so that x is_a y or x has_part y then $x \leq y$.

[3]The maintenance of the structure and integrity of the mitochondrial genome.

[4]A stable complex of proteins that carry out the DNA damage recognition and incision reactions characteristic of nucleotide excision repair.

[5]Enables the directed movement of metal ions into, out of, within or between cells.

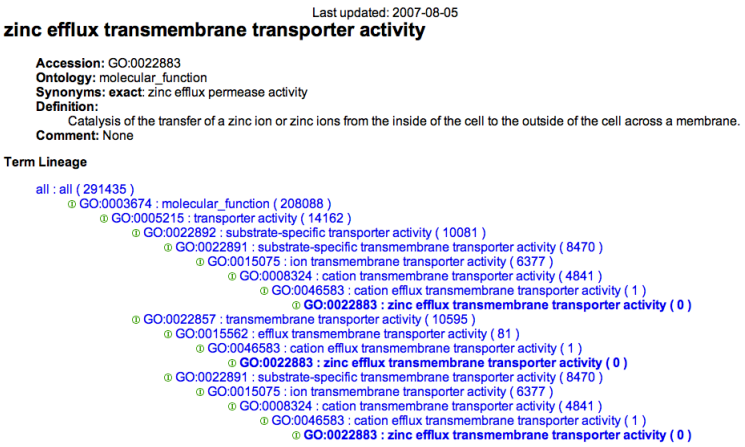


Figure 9 – *A screen shot showing a fragment of the Gene Ontology taken from the Gene Ontology browser AmiGO, showing the three different paths to the term zinc efflux transmembrane transporter activity. The numbers in the brackets signify the number of terms that exist beneath the given term. Thus 0 signifies a leaf node.*

3.1.4.2 Usage of GO

There are multiple techniques and tools that use the GO. One common usage of the Gene Ontology proceeds from a list, of perhaps thousands, of genes of interest obtained from a gene expression experiment. Then part of the approach to understanding the properties of these gene is to analyse the particular terms and their distribution within the ontology. This can then shed some light on the processes that are occurring and the functional profile of the genes [120].

The GO can also be used for querying, since every GO term can be mapped to a set of genes, the GO can be used to index into a set of preexisting experimental results - thus facilitating the exploration and navigation of experimental results. Since the GO is a species and method independent conceptualization it can be used with results from across different laboratories, systems and model organisms. For example, given a database of gene expression experimental results, a biologist may want to ask a question like: "show me all the experiments that have involved genes associated with *blood coagulation* and a cancer site." With the GO a set of genes which have some known link with *blood coagulation* can be identified. Likewise all experimental conditions associated with cancer sites can be extracted.

Returning to the specific problem that motivated this chapter: that of finding a suitable set of labels for sets (and subsets) of genes, the approach investigated here is to use the structure of the GO to aid in deciding which label should be applied to any given set of genes.

3.1.5 Annotation and curation

One of the challenges that directly affect the usefulness of the GO are the mappings or annotations from gene data-sets into the GO. One of the principal projects carrying out this task is the Gene Ontology Annotation (GOA) project.

Currently these mappings databases are incomplete and only a proper subset of known genes are functionally annotated [55]. Even when studies have shown a link between a gene and a process, the human curators of the annotation databases have not yet updated the annotation database. An example cited in [54], is of the gene HMOX2 which was shown to be involved in the process of pigment biosynthesis in 1992 [72]and is still not annotated with this GO term even today in 2007 this suggests that the curators of the GOA paid scant attention to the paper [54] or do not consider this annotation relevant.

3.1.5.1 Functional roles of genes

Associating genotype to phenotype is fraught with potential problems and mis-classifications, especially when considering higher level functions where many genes are involved in multiple processes. For example the *period* gene - originally found to play a role in the circadian rhythms of fruit-flies[56] - is also involved in timing functions such as embryonic development and in coding oscillations that are necessary for courtship. The Gene Ontology specifically limits it's own scope to exclude high level functions like the generations of circadian rhythms or embryonic development. An additional complication arises with the typical methodology used to determine the role of a gene: by differential comparison of an entity with a wild-type gene to an entity with a mutated gene. This suffers from a fundamental limitation that any behaviour common to both wild-type and mutated gene types would not be discriminated and hence it would not ordinarily be associated with the gene.

Nevertheless let a be the annotation mapping from the set of genes to the ontology, $a : U \rightarrow P$.

3.1.5.2 Process of construction

Using humans to keep annotation current is expensive and does not scale, a natural solution is to develop automated electronic annotation techniques. Indeed the vast majority of annotations are inferred exclusively by electronic annotations(i.e. without any expert human involvement). Referring to the annotations directly available from the Gene Ontology project [6] it can be seen that of the 140 000 total biological process annotations available for *Homo sapiens*, over 90 000 associations are inferred exclusively from electronic annotations. Current automated annotation techniques still have problems: annotations often may be imprecise or incorrect. The vast majority of such electronic annotations are reasonably accurate as reported by [14]. Unfortunately, some of these inferences are incorrect or are too crude:

[6] Data taken from http://www.geneontology.org/GO.current.annotations.shtml

they have been linked to high-level GO terms which limits their usefulness, in this case since depth is weakly correlated with specificity. At the present time, none of the tools allows any type of weighting (for example a p-value associated with each annotation) perhaps this may improve the management of different methodologies of annotation.

3.1.5.3 Context

User context may be the reason why the HMOX2 gene's role in pigment biosynthesis has not been considered as important as it's role in heme oxidation, since the GOA endeavours to focus on annotations that have disease relevance or are important for high throughput analysis. Many genes have multiple mappings, all current tools weight all the biological processes equally. At the moment, it is not possible to single out the more relevant one by using the context of the users' state. An example cited in [54] is that the annotations of a gene should have some ranking that is altered according to the context of the other genes in the experiment. For instance with the gene BRCA1.

> "A well known tumour suppressor but is also known to be involved in carbohydrate metabolism. If most other genes found to be changed in the current experiment are involved in processes such as DNA damage response, apoptosis, induction of apoptosis, and signal transduction, it is perhaps more likely that in this experiment BRCA1 is playing its usual tumour suppressor role. However, if most other genes are involved in carbohydrate mediated signalling, carbohydrate transport and metabolism, etc., then it is perhaps more likely that BRCA1's role in the carbohydrate metabolism is more relevant."

3.2 Ontology analysis

With the use case above of functional profiling of gene expression experiments there are often an abundance of GO terms that can be used to describe a set of genes. It is therefore necessary to find some means of expressing any particular GO terms usefulness, appropriateness or relevance. In the following subsections a methods is derived that that can be used in order to assign ontological terms to clusters of genes. The goal with this is to establish a ranking of each term when used in conjunction with the conceptual hierarchy introduced in Chapter 2. A similar form of categorization problem is posed both in Lee et al. [58] and Joslyn et al. [50] work, and in both works the tension between 'coverage' and 'specificity' is central to this class of methodologies. The significant difference that applies in the context of this work compared to the work of Lee and Joslyn: the gene clusters themselves have a partial ordering. What is thus required is then some form of homeomorphic mapping from one partially ordered structure (ontology) to another (conceptual hierarchy) whereby the ordering of the ontological terms remains invariant under this mapping.

Dedekind-Macneill completion Recalling that the GO is a partially ordered structure it is possible to add fictional elements to the poset in order to transform it to a lattice. This process called completion (specifically the Dedekind-Macneill completion [51]) is not strictly necessary for the homomorphism however it permits a definition of the height and width of the GO. [7]

The height is easily calculable, whilst a lower bound can be made for the width (calculating the width is somewhat computationally expensive). As reported in [50] on the height of the MF (Molecular Function) branch is 13 and a lower bound for the width is 3500, thus the MF is a bushy data structure which quickly branches.

3.2.1 Ordered set structural measures

Armed with this ordered set representation of the GO, considerable information can be gleaned about any GO term, by its location within the representation.

Based on ordered set theory[102] the important measurable (solely) structural properties of each term are the *specificity* or uniqueness, the *coverage* and the *centrality*. Each of these can be calculated for each term and these scores associated to each term within the GO.

The following sections describe formally and intuitively how each of the scores for these is derived.

3.2.1.1 Depth

The depth of a term is measured by the length of the shortest path from the root node to the term of interest.

$$t_{depth} = \min_{c \in I[\top, t]} |c|$$

3.2.1.2 Specificity and centrality

The *specificity* of a term is measured by the length of the path from a term to the nearest leaf node in the ontology. Therefore the shortest chain from any node $t \in S$ to the bottom node will produce the specificity:

$$t_{spec} = \min_{c \in [t, \perp]} |c|$$

The specificity for a number of elements from the Gene Ontology is marked on the figure... Counter intuitively by this measure terms which have a low specificity score are more specific then terms with a high specificity score.

[7] Recalling the definition of height (width) of lattice - in section 2.2.2.1- is the longest chain (anti chain) in the lattice.

In conjunction with the depth the *centrality* is the height of the largest chain from top to bottom that the node is a member of, for $t \in S$:

$$t_{centrality} = \max_{c \in I[\top,\bot]} |c|$$

3.2.1.3 Coverage

The *coverage* is the number of gene products that are annotated to a term so for a node $t \in S$:

$$t_{coverage} = |\{g \in U : g = a^{-1}(x), x \leq t\}|$$

The coverage for a part of the GO can be seen by the number in the brackets after each term in the screen shot from AmiGo in figure 3.1.4.1.

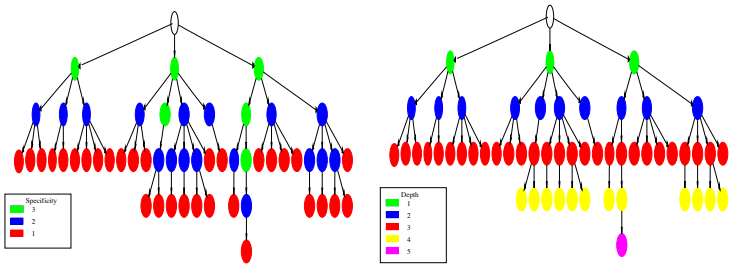
3.2.2 Informational value

3.2.2.1 Metric

The informational value metric accounts for the differing frequency of usage of terms within the ontology. By using the standard argument of information theory by Shannon, the information content of each term is defined as the negative of the log likelihood of the term being used, $-\log p(t)$, where $p(t)$ is the probability of encountering the term t. The probability of encountering a term can be inferred (in the case of the Gene Ontology) from the number of genes that are annotated to that term or any of it's descendant terms.

For example, in figure 3.1.4.1 each of the terms has the number of annotations that have been made to it from the Gene Ontology Annotation (GOA). The number of counts includes annotations made to any of the ancestor terms.

Substrate-specific transmembrane transporter activity (GO ID: 22891) has less information content than *cation efflux transmembrane transporter activity* (GO ID: 46583) which in turn has less information content then *zinc efflux transmembrane transporter activity* (GO



(a) *Specificty* **(b)** *Depth*

Figure 10 – *A colour coded diagram showing the specificity and depth for a simple tree.*

ID: 22883). This is then reflected by the probability of encountering the terms. So the p(*substrate-specific transmembrane transporter activity*) is much greater than encountering the probability of p(*zinc efflux transmembrane transporter activity*). Consequently, *zinc efflux transmembrane transporter activity* contains more information, i.e. is more specific than *substrate-specific transmembrane transporter activity* as it is quantifiable, an exact form of *efflux activity*. Also, whenever the term *zinc efflux transmembrane transporter activity* is used, the concept of *substrate-specific transmembrane transporter activity* is also implied as it is a parent node of *zinc efflux transmembrane transporter activity*. As a result, Resnik[98] concluded that the higher the terms in the taxonomy tree, the less information they contain but are encountered more frequently. E.g. if we restrict attention to the MF ontology then the term *Molecular Function* should have a p(*Molecular Function*) of 1 and a information content of 0. Resnik went on to develop a model to define the similarity between two distinct terms in the hierarchy, based on the simple criteria that more similar two terms are the more information they share. Based on this a measure for semantic similarity for two terms is:

$Similarity(t_1, t_2) = max_{t \in S(t_1,t_2)}(-\log p(t))$

where t = a term and $S(t_1, t_2)$ = set of terms that subsume both t_1 and t_2 and $-\log p(t)$ = information value of a term. Then this equation simply means that we want a term, with maximum information content, that subsumes both t_1 and t_2.

3.2.2.2 Atomic symbols within GO terms

It is possible to improve upon the informational value of a GO term by looking also at the words within a GO term, since GO terms frequently contain words from parental terms. A more specific GO terms typically includes words from more abstract terms. They have a somewhat consistent morphology about their naming structure: the suffixed portion of the term name contains the more abstract or less important terms.

For example the GO term zinc efflux transmembrane transporter activity (GO ID:15562) is linked via several hops of the 'is a' relation to the GO term transmembrane transporter activity (GO ID: 15562). Looking at the sub graph of the GO from the root to zinc efflux transmembrane transporter activity it is possible to see that most of the parental terms are integrated into the final term name.

Since GO terms have multiple parents this leads to some inconsistency in the naming of terms. This is resolved in some cases by each GO term having multiple synonyms assigned to it, in this particular case *zinc efflux transmembrane transporter activity* is also named *zinc efflux permease activity* which however does not directly aid in the multiple path issue.

For the purpose of using GO terms for naming directories it would be more usable for the directory terms to not contain words already in the path, for instance the full path of a directory labelled with the term should be something like */transporter_activity/substrate-specific/transmembrane/ion/efflux/zinc*. This can be readily accomplished when labelling by removing all previously encountered atomic symbols within a path.

This observation about the morphology of GO terms is exploited in the GOPubMed project[25] which takes abstracts from pubmed, identifying the significant words within GO terms from papers and then using the GO to provide a browsable structure to query results. A similar technique can also be found to identify atomic symbols in the GOALIE project[95].

3.2.2.3 Gene cluster annotation statistical significance

To determine which term has most significance with a particular set of genes, the technique adopted by many tools[121, 27, 54] is to use a hyper geometric test. This assumes that the probability that any GO term occurs follows a hyper geometric distribution. Then a P-value is determined by comparing the probability of observing at least k genes from a cluster of n genes by chance in a category containing K genes from a genome total of N.

Given the following: the set of genes within a cluster C, the set of genes tagged with a GO term GO_i and the set of all genes G_I. Then a contingency table is generated consisting of the four disjoint sets $C \cap GO_i$, $C - GO_i$,$GO_i - C$ and $G - C - GO_i$. P-value calculation is then:

$$P = 1 - \sum_{i=0}^{k} \frac{\binom{K}{i}\binom{N-K}{n-i}}{\binom{K}{n}}$$

The test effectively tells us whether a particular Gene Ontology term has many more genes annotated to it then would be expected by chance, it also provides a ranking system for the GO terms.

3.2.3 Usage

All the foundations are in place and the method to assigning names that preserves the (assumed good) ordering of the GO is starting to become apparent- the allocation of terms should match the structural score relations stated above. That is, in the concept lattice, deeper directories should in turn have terms with a greater depth. To integrate the above scoring metrics. they need to be calculated and available for the algorithm for labelling the directory.

Querying an ontology with standard relational databases is difficult: an example of a fairly simple query in the GO that requires finding all terms related to a particular process needs an auxiliary table and a recursive statement, the difficulty is explained in section 3.3.3.

In the next section the steps necessary in order to preprocess generate metrics and make accessible the Gene Ontology are detailed.

3.3 Implementing Gene Ontology data services

3.3.1 Introduction

In order to effectively use the Gene Ontology it is necessary to store, process (section 3.3.6) and provide access (section 3.3.7) to the terms in some form of database. It would appear to be straightforward to preprocess the terms and store the data within a relational database. However, as the next section explains, using the GO with traditional relational databases is somewhat awkward. Architecturally, to make distributed components with Inferno, it was easiest to implement an in memory data services engine accessible via simple file system operations as explained in section 4.5.

3.3.2 Gene Ontology services

In order to use the Gene Ontology, two long running data services provide data about the GO - Gene Ontology Service (GOS) - and the annotation of genes to GO terms - Gene Naming Service (GNS). The GNS is further described in section 3.4.

Whilst there exists a schema and a set of methods as part of the GO that allow storing the GO in a standard relational database, the storing and retrieval of ontology data causes some difficulties both with fitting the poset structure of ontological data within this data model and setting up queries to adequately query such a database. These issues are explored in this section.

3.3.3 Taxonomies in SQL

One useful and frequently used type of query upon poset data structures (such as ontological data) is to calculate all descendants of a node. This form of query is equivalent to the computation of the transitive closure on the $\leq$ binary relation.

Within the GO both is_a and part_of are transitive relations: a transitive relation ~ is one for which if a~b and b~c then a~c. Given any binary relation ~ the transitive closure is the smallest transitive relation that contains ~. Now clearly if a is-a b and b is-a c then it should follow within GO that a is-a c. It can be proved that transitive closure is not possible in traditional Structured Query Language(SQL)[5].

This issue can easily be seen in the GO database where queries using graph traversal (hence requiring transitive closure) form a key type of query. Within the relational database representation of the GO this issue has been resolved by precomputing every graph traversal when building the database. Thus within the GO schema there is an additional term2term table that is used to store the transitive closure. Currently the GO is not updated frequently thus recalculating the term2term table is not burdensome. However, it does result in a level of obscurity and inelegance associated to some key tasks; see for example figure 3.3.3.

```
SELECT

    child.*
    FROM
    term AS parent,
    term2term,
    term AS child
    WHERE
    child.id = term2term.term1_id and
    parent.id = term2term.term2_id and
    parent.name = 'blood coagulation'
```

Figure 12 – *An SQL query statement to find every child term of blood coagulation.*

One problem with this query is that it is not obvious which query it is answering. It uses the additional term2term table and as an idiom it is not comparable to standard tasks in SQL. The structure of the query in this case has been defined by the data not by the usage of it. A full programming language in place of SQL provides the opportunity for more succinct and clear representation of such queries.

Since the Gene Ontology is not a massive data set (the source file is less then 10 Mb in size) and initially it was not known which metrics and so in turn which algorithms would need to be implemented upon it, a simple in-memory data set loader and server was written. For the query language Inferno's built in Limbo programming language (and its associated compiler and loader structure) were leveraged. The implementation details of which can be found in section 4.5.

3.3.4 Alternate database models and interfaces

The problem of storing and accessing poset data in a relational database as outlined above is inelegent because of the constraints on the set of operations that can be performed on the data in a relational data model. There are other models of database and of database interaction that adopt a navigational model, where records are stored by linking them together. In order to retrieve a particular record a "path" is traversed through the records. Recent work in navigational models has been predominantly in object orientated databases. However the navigational model should be familiar in the context of this thesis since a principle example of a navigationally based database systems is a hierarchical filesystem.

A navigational model entails an interface that requires the user to specify the steps to obtain the information that is required wheras with a declarative interface (such as SQL) the user instead describes a set of declarative constraints for the desired result set and the database decides how to obtain the information. There are advantages and disadvantages to

both approaches. Notably the SQL approach is well suited to highly structured data, simple queries and small data sets. In particular it is harder to debug where an error is in a large single block of SQL. It can be difficult to track variables and a long query is conceptually more difficult to understand then a set of smaller navigatory steps. With extremely large datasets where speed of execution is a factor in order for an SQL database to choose a desired execution plan it must be coerced via optimizer hints.

3.3.5 GOS

The Gene Ontology Service has to carry out three functions: load the data from a file, process the data by calculating certain scores and provide the data upon a query being asked of it. The data loading is carried out upon the initialization of the database (after loading data from a source file GOS exclusively works on data stored within its memory and no functionality is available to edit information back to the file).

To run queries on the GO, I created a specific set of commands that are split into navigation (based on the UNIX commands to navigate through a file system) and data retrieval in common with other language based solutions GOS provides a powerful and flexible tool. The implementation of these ideas prompted the development of my own in memory ontology data engine within the Inferno operating system environment.

3.3.6 Measure implementation

Both of the following two algorithms insert structural scores into the data structure that holds the ontology and are run prior to the ontology service being used and after the ontology has been loaded into memory.

3.3.6.1 Resnick similarity and coverage

The first metrics implemented were the Resnick similarity metric. I used a technique inspired by [58]. Where each GO term has a GO code (different from the standard GO term id) that uniquely determines the path from the root of each one of the branches of the ontology to the particular sense of a GO term. In this system every GO term sense (where every different sense corresponds to the alternate paths that can be taken from the root to reach that GO term) can be described by a 64 bit number data type, such that the least significant bits are the same for comparable terms. The total length of the common parts of two GO codes gives a metric to understand how close two terms are. Additionally the length of the GO code bit string, representing a GO term, tells how similar a term is and hence implies higher informational value. Conversely the number of GO terms that share a particular GO terms suffix allows scoring of the coverage of a GO term.

A better method to score the coverage is for each GO term, to count how many times it (and it's descendant terms) have been annotated to by the GO annotation. Any term that is

only used infrequently has a higher semantic value. This is superior partly because it uses the annotation information in addition

3.3.6.2 Specificity and depth

To calculate (and associate with each GO term) the specificity and depth (the combined score being equivalent to the centrality) scores, a recursive algorithm based on a depth first traversal of the ontology is used as illustrated in algorithms 3 and 4.

Algorithm 3: insertspecificity

Input: GO: array of GOterms

Output: specificity annotated to each node $GO_i \in GO$

```
1 forall c ∈ GOnodes do
2     if c_specificity ≡ nil then
3         c_specificity ← specificityrecursive(c) ;
4     end
5 end
```

Algorithm 4: specificityrecursive

Input: GO_i: GOterm

Output: specificity attached to $GO_i \in GO$ and children terms $\in GO$

```
1 if GO_i·children ≡ nil then
2     c
3 end
4 specificity← scoreterm(c,0) ;
5 if cspecificty < specificty then
6     ;
7 end
```

3.3.7 Query by genes

In order to use the Gene Ontology service is indexed into by the genes that constitute the attributes of the concepts. Given a set of genes (or equivalently gene probes) there generally exist a set of GO terms that annotate the given genes. Given a set of genes which of these Gene Ontology terms should be returned, what constraints should be applied to their appearance and how should the ontological terms be ranked?

3.4 GNS

The Gene Naming Service (GNS) provides the mappings from a given gene to the Gene Ontology, It currently supports three annotations (or mappings), one based on locus ID, one

based on Affymetrix's annotation files and one based on HUGO[14].

The typical kind of data that I shall need for any GO term is the hypernyms of a given GO term and the scores of a GO term.

Access to GOS and GNS is implemented using the same idioms that are used within Inferno to provide DNS (Domain Name Service), further details on the overall architecture will be discussed in Chapter 4.

3.5 Usage of the GOS

3.5.1 Initialization

As detailed in section 4.5 the Gene Ontology Service and the Gene Naming Service are started by the command styxlisten genefs and styxlisten gosfs. Their services are then accessible from `/info/gns` and `/info/gos`, by writing instructions and reading responses to these files. For convenience the services can be started when Inferno is started by placing the commands in `$home/lib/wmsetup`. Also for convenience the mechanics of writing and reading to these files can be wrapped up in a simple Inferno shell script (further details in sections A and 4.5.) which is called by using the command `g`. The service is stateful and persistent across different sessions.

3.5.2 Basic navigational commands

Recalling that the GO is a connected partially ordered data structure, an immediate and straightforward way to query this stucture is to navigate through the elements starting at the topmost root element. For GOS to facilitate this there is a present working term that is preserved across query operations. In order to change the current term the command `g ct "term id"` is used. Any term can be made the working term if its numeric term id is known, it is not necessary to start from the root and select more specific terms in the ontology. To help locate the numeric id given the name of a term the `find` command can be given a regular expression and it will return all terms that match.

Once "at" a term, information about the term, can be found by using the command `pwt`. This gives the GO id, the name of the term, the depth and the specificity (in that order). Details about linked terms that are refinements (or generalizations) are available by using the `lsd` (or `lsu`) commands respectively.

3.5.3 Advanced Usage

Whilst originally it was conceived that all interactions would be via the basic navigational commands described in the previous subsection, it quickly became apparent that the small

```
%g pwt
loaded
6 high_affinity_zinc_uptake_transporter_activity 0 7
%g find spindle
22 %
%g ct 22
% g pwt
22 mitotic_spindle_elongation
g lsu
7052 mitotic_spindle_organization_and_biogenesis 1 10
51231 spindle_elongation 1 10
%
```

Shell Interaction 1 – *Using the Gene Ontology service.*

grammar it supports would also require control structure mechanisms and variables (and possibly functions). Rather then implementing these features they were instead appropriated from the limbo compiler. The full implementation details behind how the loading and mechanics for this are explored later in section. This also provides the significant benefit of allowing the databases functionality to be extended, easy prototyping of functionality and allows a rich set of operations on data. This style of implementation offers a solution to the tension alluded to in the introduction between persistence of data in programming languages and other data stores. In the shell interaction the steps needed to modify an existing query command are sketched out.

```
% sed -n '/pwt/,/;/p' GO.b
"pwt" =>
  if (gonodes[pwd]!=nil)
    s=sys->sprint("%d %s%d%d\n", pwd, gonodes[pwd].name,
    gonodes[pwd].specificity, gonodes[pwd].depth);
% #Edit the sprint line and change it to the following
% sed -n '/pwt/,/;/p' GO.b
"pwt" =>
  if (gonodes[pwd]!=nil)
    s=sys->sprint("%d %s\n",pwd,gonodes[pwd].name);
```

Shell Interaction 2 – *The definition of the pwt command*

The particular motivation for having this system is that it allows a sophisticated algorithm to be prototyped and developed to find a label from the Gene Ontology. The requirements and implementation of which are the subject of the next section.

3.6 Concept naming: labelling directories of a file system

3.6.1 Criteria

A naive approach to assigning labels to concepts, would be to annotate the directory with the Gene Ontology term with the lowest p value calculated with the technique described in section 3.2.2.3. However there are several issues which necessitate a more sophisticated approach.

With the file system, a formal concept is transformed into multiple directories and so there are generally a multiplicity of labels that can be associated with any formal concept as explained in section 2.3.3. An alternate way of considering this, is that each cluster can have (in effect) multiple parents and in turn multiple paths from the root to the directory, and each alternate path will need to have a different set of labels attached to it.

Hence a proposed better solution can be found by considering the structure of the entire file tree and reflecting the structure of the GO. Thus in order to build this more global solution, the names that these directories take should be based on a quality function that is a heuristic based on aiding usability and findability of patterns. What this requires in addition to scoring the quality of a term, is to score or rank the directories by some usage quality. A satisfactory (or stable) global labelling exists when every term associated with a directory has no other term which has a better score to represent those genes.

The following criterion can be applied to define the heuristics of assigning names to directories:

1. Every term should be an appropriate label for the genes it contains. Thus the majority of genes should have an annotation to the term.
2. Every sub-directory should have a unique term.
3. No ontological term should appear two or more times in any path from root to a directory.
4. More general ontological terms should be used closer to the root of the file-system. More specific terms should be used in the leaves of the file-system. Such that every term in a path should be more specific then its parent terms. Thus the ordering of the GO terms should be preserved.

It is also worth noting that every sub-directory C of a directory D should contain fewer experiments or experimental conditions (objects) and more genes (attributes) and that the genes associated with a directory are a subset of the genes associated with a sub-directory: $C.experiments \subseteq D.experiments$ and $D.genes \subseteq C.genes$. This is since the experiments form the objects and the genes the attributes when building the concept lattice. Criteria 4 implies that a GO term's location within the file system is homeomorphic to its location in

the Gene Ontology and preserves the *Depth, Centrality* and *Coverage* scores(as defined in section 3.2.1). An alternate way of viewing this is to imagine the Gene Ontology being squeezed and stretched onto the file-system.

It also seems more useful if the names of the directory only contain the specific parts of the GO term name rather then the more abstract ones. This is based on the assumption that directories higher up in the file system tree have been labelled with the more abstract terms. There are generally going to be more available GO term labels then there are directories to label however the unfolded lattice reflects the tree structure of the ontology with the lower concepts having many more paths (and hence directories) that will eventually need to be labelled compared to the higher concepts.

3.6.2 Algorithm implementation

In the previous chapter in section 2.3 the next closure algorithm was described, in particular, its property of iterating through the concepts starting at the bottom and ascending through the levels of the concept lattice as it finds concepts. This useful property is exploited in the first part of the ontology naming process. A findterms function is called on each concept as the next closure algorithm ascends the concept lattice.

The findterms function, first assigns for each concept C_i a set of candidate ontological terms T_i. These terms are obtained from the annotation mappings for each gene, typically each gene will map to several GO terms. Each term T_{ij} is scored by the percentage of genes which are annotated to it or one of its descendant 'is a' terms. Thus if there are two genes belonging to a cluster C_i and the term *efflux permease activity* is annotated to a gene, and the term *zinc efflux permease activity* is annotated to another gene then *efflux permease activity* would get 100% while *zinc permease activity* would only score 50%. Next, divide the coverage with the structural coverage score from section 3.2.1.3. The terms can now be sorted by their scaled coverage.

Next, for each level of the concept lattice the depth threshold value is fixed and all terms must be equal to or greater then the given depth. As the algorithm proceeds up through the levels of the lattice the depth threshold increases. Now for each concept there are a set of ontology terms, ready for the next stage in the algorithm where the lattice is unfolded.

Algorithm 5: Labeller in conceptderive

Input: C, concept lattice C with n elements $C_0 = \top$ and $C_n = \bot$

Output: $G_i \subseteq G$ as putative terms for each C_i

`/* lookupthreshold finds highest and lowest terms in GO. */`

$(GOdepth_{min}, GOdepth_{max}) \leftarrow$ lookupthreshold($C_n.genes$) ;

$Cdepth_{max} \leftarrow \max_{\forall c \in C} c.depth$;

$floor \leftarrow GOdepth_{min}$;

$ratio \leftarrow (GOdepth_{max} - GOdepth_{min})/Cdepth_{max}$;

$C_n.terms \leftarrow findterms(c_i.genes, floor)$;

forall $d \in C_n.superconcepts$ **do**

  $d.terms \leftarrow labelrecursive(d, ratio, floor)$;

end

Algorithm 6: labelrecursive

Input: C_i,ratio,floor

Output: $C_i.terms$, Set of putative terms labelling for each i node and it's parental nodes

if $C_i.terms \equiv nil$ **then**

  $C_i.termsi \leftarrow lookupterms$;

  forall $c \in C_i.superconcept$ **do**

    labelrecursive(c,ratio,floor+ratio)

  end

end

3.6.3 Unfolding the lattice

As the lattice is unfolded to create the file system, it uses a recursive depth first graph traversal over the concept lattice. At this stage all concepts have several putative terms allocated and this next part of the algorithm needs to determine which terms and their atomic symbols are most suitable for the current path. Ideally most available terms should be used in some part of the file system hierarchy.

This second part of the labeller algorithm works backwards from the first part it follows every path down adding labels to the directories. It is constrained to not use terms which are more abstract further down the path and not to use atomic symbols that already appear in the path.

Input: Concept Lattice Elements, C_i,path,specificitythreshold
Output: A label for each C_i

forall $t \in C_i.terms$ **do**
  if $t.specificity > specificitythreshold \wedge t \notin termsused$ **then**
    break ;
  end
end
if $t \equiv nil$ **then**
  return $C_i.attributes$
end
$u \leftarrow$ tokenize(t) ;
forall $x \in u$ **do**
  if $x \notin stoplist \wedge x \notin path$ **then**
    $path+ = x$;
    $name \leftarrow x$
  end
end
return $name$

3.7 Improvements to ontology

This section is devoted to discussing the nature of pragmatic ontological structures, by analogy to other fields that have used categorical knowledge management structures. It is worthwhile considering which ideas have (and have not) found usage and application in these other fields to infer what is likely to be successful within biological knowledge management projects. Certain semantic web ideas have impacted how people use the Internet now, however the form or implementation of these ideas has come about via radically different means. By analogy it is possible to infer where biological knowledge management is going. In addition there are some suggested future directions that ontology can go.

3.7.1 Improvements and evolutions to the Gene Ontology

The Gene Ontology is actually quite a primitive data structure. There are some immediate and likely enhancements that the Gene Ontology may end up incorporating. This section describes these improvements and identifies how this may affect the other work.

3.7.1.1 Relational biology

As it stands the Gene Ontology only supports two binary relationships between terms. In comparison other ontologies including Cyc Sumo[80] and WordNet [30]have a plethora of available relationships between terms.

Within the paper by Barry Smith et al[105] a methodology is advanced for providing a set of formal definitions for the relational expressions and extending the relations used in Biological Ontologies. Subsequent to the paper the ontology of relations [8] has been updated and currently it consists of the following 25 relations:

- Foundational: is_a, part_of, integral_part_of, proper_part_of,improper_part_of, instance_of
- Reciprocal Foundational: has_part, has_integral_part, has_proper_part, has_improper_part
- Spatial (connecting one entity to another in terms of relations between the spatial regions they occupy): located_in, contained_in, adjacent_to
- Reciprocal Spatial: location_of, contains
- Temporal (connecting entities existing at different times): transformation_of, derives_from, preceded_by
- Reciprocal Temporal: transformation_into, derived_into, precedes
- Participation (connecting processes to their bearers): has_participant, has_agent,
- Reciprocal Participation: participates_in, agent_in

Of which the Gene Ontology currently uses two: is_a and part_of. What has been highlighted as one of the strengths of the GO also indicates a potential source of issues for the GO; the crudeness of available relations between terms. Indeed much of the multiple inheritance in GO - as presently constructed - which is often associated with errors in ontology construction mainly because it demonstrates the (ab)use of the 'is_a' relationship for multiple different meanings. For instance the 'is_a' relationship is used: '*extracellular region is_a cellular component'*. This is ludicrous since the extracellular region refers to the space outside the cell.

Ontologies have historically had the property of single placement within a hierarchy, (as covered by [84]). However as explained by Morville[74]. Contextual issues arise from these methods of categorising, many objects do fit within multiple categories, however if it is not necessary multiple inheritance should arguably be avoided.

A further exposition into the relevance of this upon the representation of complexity in biological systems is saved for a later chapter. However at this point it is worth briefly noting

[8] It can be found here http://obofoundry.org/ro/

that category theory has been suggested as a framework for capturing heterogeneous relations between ontology terms [52].

One of the hierarchies in the Gene Ontology is Biological Process however there is as yet no well defined annotation between the temporal description of behaviour of a Genetic network and its Gene Ontology term.

3.7.1.2 Annotation and folksonomies

Earlier in this chapter some of the issues within the annotation process (mapping genes to the Gene Ontology) were identified. One of the key bottlenecks in the process of annotation is the time it takes for librarians and ontologists to mark up or categorise data. How can the creation of meta data be improved to cope with ever increasing volume and complexity of data?

There have been attempts to improve automatic annotation and text mining in biology. Electronic annotation can be likened to machine translation of human languages. Recently machine translation has improved by a significant amount, at least part of this success is beyond superior algorithms and techniques rather the increasing volume of training material available. This can actually be observed by considering the automated Google translation of "The flesh is willing but the spirit is weak" in English to Japanese to English. This translates to "The meat is rejoicing, but mind is weak" however the English to Chinese to English gives the same result as the input. One of the key differences between the two languages is the quantity of training material available for both data sets, since Google Translate works by:

> "...we feed the computer billions of words of text, both monolingual text in the target language, and aligned text consisting of examples of human translations between the languages. We then apply statistical learning techniques to build a translation model. We've achieved very good results in research evaluations." [9]

One approach suggested in [23] is to let authors explicitly mark up their findings and claims in a semantic form suitable for search engines. An alternative to a mark up scheme is a dynamic automated gene to term weighting by auction system linked to the impact factor associated with a publication. The rationale for this is based on the observation that some of the commercial aims of the semantic web are closer to being brought to fulfilment by Google Adwords[10], rather then increased semantic mark up by authors.

Taking this beyond the annotation process a similiar procedure could be used to create emergent categorisations that are defined from the data rather then adopting a top down

[9] From http://www.google.com/intl/en/help/faq_translation.html

[10] Google Adwords (https://adwords.google.com) is a keyword based advertising system for the Google search engine. Advertisers bid on keywords, adverts are then displayed in a particular ranking to the side of search results when the corresponding keywords are entered in a search query. Advertisers only pay when their adverts are accessed. Advertisers can improve the ranking of their adverts by either paying more per an access or increasing the frequency their adverts are accessed.

approach to categorisation. The end result is a data structure that is more like a so called folksonomy[11] in comparison to the Gene Ontology's taxonomy. In a metaphor credited to David Weinberger[74] discussing the difference between folksonomies and taxonomic structures: "The old way creates a tree. The new rakes leaves together." It should be made clear that a scheme like this acts as a supplement not a replacement to the Gene Ontology and its annotations, since its data won't be as thorougly verified. However it could still add value by providing serendipitous extra meta data and providing a mechanism for faster dissemination of research knowledge. In the system described in chapter 4 within this thesis the methodology to extract a concept lattice could be applied against multiple datasets and the results integrated to form exactly such a structure.

3.8 Summary

In this chapter I have shown how an ontology can be treated mathematically. I demonstrated some of the fundamental operations that are useful when engaging with an ontology and introduced a novel data engine that calculates and provides information from the Gene Ontology. The major novel contribution of this chapter is an algorithm for using words (as atomic symbols) from the Gene Ontology to label concept directories derived from gene expression experiments. The major difference between this system and preexisting tools that use ontology and annotation is the effect of users context in which terms in the Gene Ontology are used.

I suggested some extensions and future directions that the Gene Ontology might take.

Finally I argued, by analogy with other applications of ontology or meta data, that hand curated ontologies as a tool may yet be improved by emergent methods based on aggregating large volumes of data.

In this chapter I described the fundamental algorithms and data structures that form the basis for an ontology storage layer. In the next chapter I shall describe a Gene Ontology tool that implements these ideas. Later in chapter 5 I shall describe how this component works together with other tools to construct a system to explore and analyse data in an easy to access format: a file system.

[11] A system of classification based on collaboratively creating and managing tags.

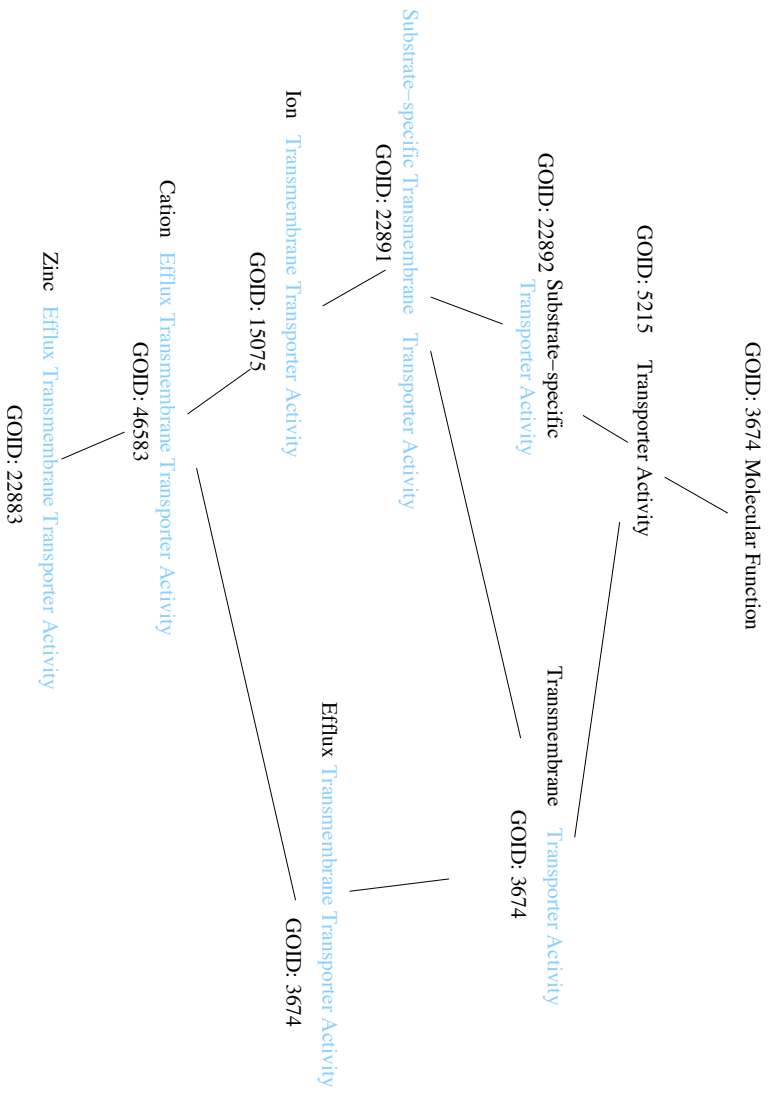


Figure 11 – *A subset of the Gene Ontology focusing on the term zinc efflux transmembrane transporter activity. Words that have already been used in a term higher in the Gene Ontology are coloured blue.*

Chapter 4

Formal concept file system

In this chapter the ideas discussed in the previous two chapters are reified towards a working system for biological data management.

There is a powerful design pattern that has been used within the novel research operating systems Inferno and Plan 9 of consistently representing every data structure and resource with a file or a file-server. Many of the design decisions and challenges about how to wrap a resource or a data structure as a file-system have answers within the Inferno and Plan 9 systems.

The following section focuses on Inferno and explores how the file-system forms the basis for almost all resources within the system. A major issue of *how* Inferno has been implemented has been the decisions that were made regarding the representation of data structures by file systems; this is primarily been driven by the context and expected usage. From these design designs a summary is made of the heuristics for effective representation of resources by file systems.

There have been other attempts to build Formal Concept File-systems, brief reviews are carried out of two recently developed and experimented efforts: LISFS and libferris.

From this two different types of file system architecture through which knowledge resources are made available: the Formal Concept File-system and service oriented programs exposed as files.

4.1 Introduction to Inferno

4.1.1 Inferno and Plan 9

Inferno and Plan 9 are operating systems much like Windows or Linux, however they were cleanly designed to fully integrate and exploit networked environments and address the problems of distributed computing[91, 43]. Inferno is in some ways the successor to Plan 9 and builds on its protocols and methodology of Plan 9 which are in turn is both an extension of

the original ideas and response to the weaknesses of UNIX. The underlying philosophy has been to try to always solve the most general problem by using the simplest solution[90, 53]. Infernos features are summarised below:-

- Remote Procedure Call mechanism packaged within a straightforward file-system protocol - Styx
- Resources represented as files
- Trivial network access to resources
- Lightweight and elegant virtual machine architecture (dis) with built in garbage collection
- All applications developed in a type and memory safe language
- Modern, easy to use, concurrency implementation based on the theory of Communicating Sequential Processes (CSP)
- A simple typed work flow framework
- Inherently Grid-enabled architecture

Despite this Inferno has never achieved mainstream usage only gaining a rather small niche of users. The reasons for its lack of success are predominantly non-technical: it came to market approximately one year after it's nearest competitor (Java) it never achieved a critical mass of users and was poorly marketed.

4.1.2 Architecture

Inferno runs upon a virtual machine, and the system has been ported to multiple hardware architectures.[90] Additionally it can be run from within another OS as an application in a similar manner to the Java virtual machine architecture[116]. Currently actively supported platforms include Windows XP (and all related systems based on Win NT), OS X, Linux, FreeBSD and Plan9.

One of the key features of the Inferno architecture, is the representation of all resources as file systems. The operation of the system is then due to the interaction of each of the file servers together with various kinds of input to the system. Inferno was built from the ground up to easily support networking and running as a distributed system, and carries with it 20 years of research into distributed systems from Lucent Technology (formerly Bell Labs). It uses its own programming language - Limbo, intended for applications running distributed systems.

4.1.3 Limbo

Limbo - Inferno's native programming language - like Java has strong type checking, automatic garbage collection and safe execution (it does not allow potentially unsafe pointers). It's syntax and expression has links with the newsqueak language[87] (it's predecessor) and with sawzall [89](it's successor.)

All applications and many system modules in Inferno are developed in Limbo and when run hosted its performance is comparable to other interpreted languages and with some care it can be made to run well [40]. Unlike Java it is not object oriented, though it has a modular architecture. Programs developed in Limbo are compiled into a platform independent bytecode. Thus any Inferno application should work the same on every platform that is supported by Inferno without needing porting or even recompiling.

Limbo has a syntax that is similar to C, see [26] for a full introduction to the language. It has some key general benefits over C (C is the more usual language for systems level programming) including support for generic containers and a basic string class and specifically for systems level programming has some security advantages over ANSI C, due to it having strong type-checking and an inbuilt garbage collector.

The design and development of file servers within Inferno is greatly facilitated by the concurrency model underlying the Limbo language. Further details can be found in A.3.

4.1.4 Styx: a common network access protocol

The resources accessible to an individual namespace can be located on a single machine or on multiple machines throughout the network. This has multiple benefits: computer users and programmers are familiar with files and their semantics, a set of resources can easily be distributed around a network and it allows the same access protection that exists for normal files to apply for all resources. Wrapping operations by an interface that is a file server has several benefits in implementation and in use, mainly that many existing tools can easily be used with that resource.

Inferno implements an open protocol, Styx for handling file operations (the following is mainly summarized from [94]). Styx is analogous to Sun's NFS[73] or Microsoft's CIFS. It does not directly implement caching strategies to optimise regular disk file access, and hence is somewhat slower then other protocols. However this makes it far simpler and its behaviour with dynamic file systems is better defined. Every file operation can be translated into a sequence of Styx messages. In essence the Styx protocol is the specification of the exchange of messages. It is a stateful communication and provides a hierarchical file system name space. The process is somewhat different to how the Hyper Text Transfer Protocol (more commonly referred to as http) works. Whereas in the web at every interaction there is no retained state and mechanisms such as cookies need to be used, a Styx client retains the state of an interaction as a file handle. Every interaction after the initial attach specifies the

handle that is being acted on.

The Styx protocol is synchronous and consists of 13 message types. Every client of the file system maintains a handle to whichever part of the file system is being accessed.

Table 11 – *Styx messages*

Name	Description
`attach`	Authenticates client and provides a handle to the root of the server tree.
`clone`	Provides a new handle to the same file.
`walk`	Point the file handle to a specified next level of the file hierarchy.
`open`	Check permissions for file I/O.
`read`	Read contents of file
`write`	Write contents of file
`flush`	Disregard outstanding I/O requests
`stat`	Report meta data about a file
`wstat`	Change meta data about a file
`close`	Discard file handle (file itself is unaffected)
`remove`	Delete file
`create`	Create file

Typically one process (the file server) accepts requests for files and file structures and carries out the commands associated with those requests. The mechanism of a client server interaction is sketched out: firstly a client attaches itself to the root of the server. The client process is given a file reference number of the root of the file server. When the path of a particular file from the server is requested the client walks from the root to where the file is. Within the server every file has a uniquely assigned number (called a qid). Further details can be found in subsection 4.6.

Styx messages do not need the existence of a network to join two components, Styx can be used over shared memory or UNIX pipes. Importantly its behaviour joining a local component is identical to joining a remote component. This means that remotely mounting parts of the file system of anything that talks Styx gives corresponding control of that remote system. A distributed system can therefore easily be assembled by mounting the required services and defining a minimum of code to govern the interactions, without having to give much consideration to the underlying hardware or network connection.

Within the Inferno system, the only requirement for any process to become a user file system is that it has to read messages in the Sytx protocol over standard input and produce Styx messages over standard output for it to be mountable within the file system.

For example the helloworld program listed below (with source code in the appendix), connects Styx messages sent to it's standard input and is used like this:

This server only serves one file, called `hello` and reading from that file returns the string world, all other styx operations are ignored. For another example see section 4.4.3.

To export a portion of a namespace over the network the `styxlisten` command is used,

```
% mount {helloworld} /n/test
% cat /n/test/hello
world%
```

Shell Interaction 3 – *Mounting and using a user file server*

```
% echo $emuhost
MacOSX
% styxlisten tcp!*!1278 export '#^'
% echo hello | os pbcopy
% cpu romulus
% echo $emuhost
Nt
% mount tcp!whitegene!1278 /n/remote
% ls /n/remote/
snarf
% cat /n/remote/snarf
hello%
```

Shell Interaction 4 – *Sharing the clipboard. Here hosted Inferno running on a OS X machine (whitegene) exports its clipboard to hosted Inferno running on a Windows machine (romulus). Text copied in whitegene can be accessed in Inferno on romulus.*

Styx and the philosophy of writing file servers can be used as middle-ware[13], this is "glue-like" software that links different software components together. In comparison to approaches like CORBY or SOAP it has the advantage that clients do not require special language bindings (clients only need to be able to access files) However it does need to run on a system that permits a Styx based file system to be mounted. Currently Linux and OS X have buggy support for mounting Styx servers. There exist libraries that permit Styx based servers to be written in Java and Python.

4.1.5 Per process file system based interface - computable namespaces

Superficially Inferno is similar to *nix systems (like Gnu/Linux or FreeBSD) with many minor differences but one important and key evolution: everything is represented and controllable within the root file structure by actions on files. The hierarchical file tree does not just represent data on the disk but can be interfaces to networks, protocols, the graphics system, audio, security and remote execution.

Opening, closing reading and writing of the appropriate file is the interface for virtually all parts of the system. This has the effect of heavily reducing the number of commands needed for the system, it facilitates programming and promotes code reuse. Files become an abstraction and can be thought of as *names*. The file system, which is the interface to most system resources, is called a *namespace*, a resource then only has to be attached to the namespace for it to be accessible. Once a resource is attached to the namespace it is able to interpret all file systems accesses and synthesise files and directories on demand.

There is a bootstrapping problem, where do the initial namespace components come from? Each kernel device drivers is a file server and is accessible via a unique # prefixed character. Thus each driver can be bound into the namespace by an operation as seen in shell interaction 5:

```
% bind -a '#p' /prog
```

Shell Interaction 5 – *Binding the program control scheduler into the prog directory.*

Conceptually all device drivers adhere to a simple format of providing the necessary functions for a styx interface and hence the kernel can easily plug in file system accesses to the device driver. There is still potentially a great deal of freedom in how a device then appears on the file system however there are some conventions as to the structure of the file tree and the naming of files. There are some commonly occurring files that are needed by many file servers:

- `ctl` - Control file, instructions to the file server should be sent here, this is used instead of the ioctl system call in UNIX.
- `clone` - New allocations of resources should occur with opens of this file, rather then utilising the create file mechanism. This permits the file server to have control over naming and reuse of files. An example of this can be seen in shell interaction 8 where to get a new connection the clone file is opened.
- `status` - Provide the status of components
- `data` - Reading will return any data provided by file service
- `1,2...` (numbers) - List of some allocated resources all of which share some commonality, they may be reusable

Each running process (or thread) in Inferno has its own view of the namespace, by which it can have a unique and private view of the resources and services that it needs to access. This view is easily manipulable and no special privileges are needed to modify this view since it is not generally globally applied. This is unlike on *nix systems where mounting a volume generally requires root access. Every new process that is created inherits its namespace from its parent creating process.

It is worth clarifying that while a resource is represented by a file, not all file operations are meaningful or defined upon it. Files representing resources may be read-only and moving file representing resources may not be possible. On the other hand the semantics of file operations correspond well to some kinds of interactions for example the clipboard is a file `/dev/snarf`, that can be read to obtain the contents of the clipboard.[1]

[1]Furthermore on hosted Inferno the `/dev/snarf` file is provided by a device (`#^`) linked to the underlying operating systems clipboard.

In some places the representation of a resource by a file is not useful and it is worth extracting the heuristics adopted by the developers of Inferno and Plan 9. In this section 4.1.5.1 I explore how the file-system metaphor is used to expose the networking resource in Inferno.

4.1.5.1 An example file server: the network device

Inferno (and Plan 9) adopt a novel approach to the representation of a network device and the protocols that it supports, I go through this implementation to see the philosophy and design approaches that were adopted since much of this informs the decisions made when implementing our resources as file-systems. The implementation of the network device is similar to other devices on the system more importantly it illustrates exactly where and how 'the everything is a file' philosophy should be applied. Much of the following is adapted from [75] by Sape Mullender and Dave Presotto.

There are multiple ways that networking may be available to an instance of Inferno, depending upon whether it is running native or hosted and in turn which platform it is being hosted upon. Rather then having multiple context dependent ways of interacting with networking, it is instead abstracted and wrapped in an interface that is a set of files served by a file server and mounted in the namespace.

Networking is then accessed and controlled by opening, reading and writing files. If it is running native then the networking directly interacts with the networking device on the other hand if it is running hosted then it instead calls the host networking API, for applications running within Inferno there is no difference in operations. I examine how the system represents the network interface by looking at the file system which is conventionally mounted in `/net`.

```
% cd /net
% ls -l
--rw-rw-r-- I 0 network abhey 0 Dec 19 19:43 arp
--rw-rw---- s 0 abhey   abhey 0 Dec 19 19:43 cs
--rw-rw-rw- s 0 abhey   abhey 0 Dec 19 19:43 dns
--rw-rw-r-- I 0 network abhey 0 Dec 19 19:43 ndb
d-r-xr-xr-x I 0 network abhey 0 Dec 19 19:43 tcp
d-r-xr-xr-x I 0 network abhey 0 Dec 19 19:43 udp
%
```

Shell Interaction 6 – *The /net directory*

There are four files and two directories. Each of the files and directories are actually interfaces to a running process or a device and the character after the user permissions in the above listing (in the second column and which are I or s above) signifies which device is providing

this file or directory. Each protocol (`tcp` and `udp`) has its own directory, within this directory there exists a `clone` file, a `stats` file and sub-directories numbered from zero to the number of connections currently configured for this protocol.

```
%cd /net/tcp
%ls -l
d-r-xr-xr-x I 0 abhey   abhey 0 Dec 19 19:43 0
d-r-xr-xr-x I 0 network abhey 0 Dec 19 19:43 1
d-r-xr-xr-x I 0 network abhey 0 Dec 19 19:43 2
d-r-xr-xr-x I 0 network abhey 0 Dec 19 19:43 3
d-r-xr-xr-x I 0 network abhey 0 Dec 19 19:43 4
d-r-xr-xr-x I 0 network abhey 0 Dec 19 19:43 5
d-r-xr-xr-x I 0 network abhey 0 Dec 19 19:43 6
--rw-rw-rw- I 0 network abhey 0 Dec 19 19:43 clone
--r--r--r-- I 0 network abhey 0 Dec 19 19:43 stats
```

Shell Interaction 7 – *Network indexes each connection by number*

To use a particular protocol to make an outgoing connection, the following method is used. Within each protocol directory, opening the `clone` file reserves a connection, and the resulting file descriptor is open on a `ctl` file for that connection. If there are unused connections then opening the `clone` file reserves a free connection otherwise a connection directory is created.

Network access using the tcp protocol would be initiated by opening the file `/net/tcp/clone`. This returns a file handle to a directory for example `/net/tcp/6`. Now read and writes to the files within this directory are the interface to this particular connection. In then copying the file a bind can be carried out to make this resource available at a place in the namespace.

Each connection is represented by a numbered sub-directory, reading the `ctl` file returns the number of the directory representing the connection. Writing strings to the `ctl` file controls the usage of that connection. For example shell interaction 8 shows an http connection being made to google. During the whole interaction only `open`s, `read`s and `write`s to files are necessary to use the network interface. Knowing the semantics of which set of file operations to use is however non-trivial (and there are helper libraries that encapsulate frequently executed operations).

The directory structure and the naming of the files mirrors the data structure that exists within the running network file server.

Some details are worth noting, each connection could have been named using the name of the server being connected, to rather then a number. For example the connection could have been named "/net/tcp/google" instead of "/net/tcp/1". However in this case, operations like reusing a connection or querying over all connections would not be so easily defined. A stack of connections becomes a list of numbered directories. New files are never created directly by the user, the file system is kept read only. Similarly while it would have been possible to

```
% cd /net/tcp
% cat clone
1% cd 1
% ls
ctl
data
listen
local
remote
status
% cat ctl
1
%<> /net/dns {echo -n google.co.uk ip > [1=0];
  addr=`{read};
  echo $addr
}
72.14.221.104
%<>/net/tcp/clone{connection=`{cat}
  echo connect 72.14.221.104!80 >[1=0]
  cd /net/tcp/$connection
  cat data &
  echo GET index.html >data
  echo >data
  echo >data
}
%
```

Shell Interaction 8 – *Connecting to google using only cat and echo.*

assign a separate directory for each port, splitting the underlying resource in that way would not have helped most uses.

The power of the consistent file system metaphor can be seen from the straightforward ability to do the following: import the network stack from a gateway computer, make a command look like another network connection (see execnet on Plan 9), or restrict access to the network.

4.1.5.2 An example file server: the domain name server

The file server interface provides a uniform and seamless interface to networking functionality. There still however lurk fiddly details. One of which is the Domain Name Service (DNS). DNS is the means by which easy to remember alphabetic Internet domain names are mapped to the actual underlying numerical IP addresses of the Internet.

Each one of the protocols supported by Inferno uses a different addressing scheme. Thus the control strings that are written to the `ctl` file need to be different depending upon which protocol is being used (though the most common application is translating from alphabetic names to numerical IP addresses.) One possible solution that was considered and rejected (and which at first glance seems a good idea) is to have a user level file system that represents the network name space as a file tree. The file hierarchy then provides paths to directories representing network domains. Each directory contains files representing the names of the machines in that domain. For example there might be a file /net/name/uk/ac/york/romulus. The contents of this file would be information about the machine romulus, like its IP address. This implementation was not eventually used and the following reasons are cited [76]:

1. It was found difficult to devise a hierarchy that encompassed all representations of addressing schemes in a coherent manner.

2. The address of a machine is only a part of the information that is required to make a connection. For example with IP protocols the symbolic to numeric port number mapping is required but it is not clear how to encode that within a file operation.

3. Networks have large numbers of machines and placing all these nodes in the file tree burdens the user with a great deal of unnecessary information about the organization of the network.

It was found in this case the representation of a resource by a file tree was inappropriate. The method used instead in Inferno to carry out DNS queries is to have a DNS server process which serves a single file (`/net/dns`). Then to make a query, it is necessary to open a file write a string representing the query into the file. Reading the file returns the query result.

```
<> /net/dns {echo -n 'google.com ip' >[1=0];cat}
google.com ip 64.233.187.99
google.com ip 72.14.207.99
google.com ip 64.233.167.99
```

Shell Interaction 9 – *Accessing the DNS file-service*

4.1.5.3 HURD: Alternate approach to user level program file systems

The "Herd of Unix-Replacing Daemons" (HURD) is a novel implementation of a UNIXkernel like environment that consists of a set of servers running on the Mach 3.0 kernel[110]. Its approach to file systems is similar to Inferno; a path-name on the namespace can be attached to a user level program (referred to as a translator) and which will perform specialised services when that path-name is accessed. Its approach to a client application like ftp is worth comparing to the Plan 9 way (I use Plan 9 since Inferno does not come with a corresponding ftp client). In HURD to access the ftp client the user changes directory to the directory /ftp/prep.ai.mit.edu/pub/gnu. Whereas in Plan 9 the command ftpfs[2] prep.ai.mit.edu is first issued then ftp access is provided via a file server at /n/ftp. Within HURD changing to a directory may carry out an operation whereas in Inferno changing directories is not typically a method of controlling a given resource. [2]

4.1.6 Heuristics of resources as file servers

What features of a resource have to match the paradigm of a hierarchical namespace for there to be benefit in representing it as such? What are the idioms that are well defined and adhered to? Whilst there is copious documentation regarding the Plan 9 System these heuristics have never really been fully codified. The following is the author's attempt to provide some general hints taken from the implementation of resources including network, cameras, task control and grid scheduling have been implemented via the file interface. Most are derived from the source code and from the following Plan 9 and Inferno papers[92, 43, 88, 93, 91, 32, 75].

1. Only use as many files/entries in the namespace as you need.
2. The directory structure should reflect the structure of the underlying data model of the resource. Not every data structure within a resource should be split over a hierarchical resource.
3. File system level commands should have their normal behaviour considered first before overriding to create new functionality. For example if some control is wanted over the allocation of new resources then use a `clone` file mechanism rather then requiring the creation of a new file.

[2]There is one notable exceptional example of when changing directories invokes a command, and that is the mntgen command. Mntgen serves styx messages and generates sub-directories on demand, and is typically used to provide mount points on demand for services or file-systems.

4. Controlling a resource should be done via writes to a `ctl` file.

5. Changing to a directory should not invoke functionality. (compare with HURD)

6. A textual interface that permits as much functionality to come from existing UNIX (and hence tested and well known tools) rather then introducing potential bugs with new software.

7. Some resources should not be represented by a file system, where it would lead to remote behaviour that is inconsistent with reality. For example shared memory on Plan 9, in this case to do so would imply that memory can be imported from remote machines.

4.2 Review of semantic file systems based on FCA

The notion of a *semantic file system* was introduced by Jouvelot in [22]. It is defined as a virtual arrangement of directories that allows flexible associative access to the contents of files by automatically extracting attributes from files with transducers. Transducers are user defined programs that can derive attributes from a file. These attributes are then used to build corresponding indexes. This generalises the model commonly used in file-systems where only a fixed set of attributes (time of modification, user permissions, size of file etc) are available for each file. In semantic file systems, the attributes are dynamic entities that can be created and accessed at will. Semantic file systems are thus more generic examples of file systems.

The motivation to base a semantic file system on FCA comes from the ability for the system to handle under-specified queries, the provision of an ordered grouping on query results and the ability to switch between query and navigation. Other semantic file-systems like SFS[22], HAC[39] and BeFS[37] for example do not permit the navigation of arbitrary queries.

4.2.1 Libferris

The libferris[69] is a virtual file-system that provides the ability to treat many tree like data structures as file-systems and present the main content of each object in the tree via a file interface. Like many Plan 9 file-systems its file-systems runs in user address space.

It has support for mounting a multitude of entities including but not limited to Evolution email clients,XML files, PostgreSQL databases, tar files and websites. Libferris also provides mechanisms to extract meta data from the input object and represent this meta data as extended attributes of the file associated with that object. Thus a picture file might have the width and height associated with it, represented as a pair of extended attributes.

To take advantage of libferris requires applications that are written using the libferris library. Hence it includes rewritten versions of basic UNIX programs like ls, cp, mv, rm, mkdir, cat, find, touch and IO redirection.

Wherever possible the structure of the file-system is taken from the entity being mounted thus for XML files it uses the Document Object Model to represent the file as a tree of objects, from this tree a hierarchical file-system is assembled.

The principal current FCA usage within libferris is to use the file-system and filter over the attributes to generate formal contexts (or binary matrices). Since libferris permits multiple data-sources to act as file-systems, this in turn facilitates multiple ways of obtaining source data for FCA. There are some significant philosophical difference worth drawing out within the implementation of libferris compared to Plan 9, in libferris a directory is a container with iterators in libferris. In Plan 9 the file system is a device that 'speaks' a networkable file protocol, hence a libferris file-system as implemented only works upon a single computer.

It can currently only be used in GNU/Linux.

4.2.1.1 Libferris example

Formal Concept Analysis can be used by libferris on any file-system that libferris can represent as a file-system. Each file-system in libferris is accessed via a url based namespace, so for example to access a PostgreSQL database called play the command `ferrisls pg://localhost/play` can be used.

To use FCA in libferris first requires building an extended attribute index of files. Boolean queries are then used to define a binary context table which is then built from the extended attributes.

```
% ferris-create-fca-tree --index-path=EAPATH --treename=testB \
smallsize '(size<=100)' \
gnomename '(url=~.*gno.*)'
```

Shell Interaction 10 – *The ferris-create-fca-tree command builds a lattice, with two attributes - smallsize and gnomename.*

In figure 10the attributes are defined by two Boolean queries. Calculating these Boolean queries over the files (object) then produces a binary matrix or context table which is used to produce a concept lattice file-system. From the binary table a concept lattice is created and is accessible from the mount point fca://.

4.2.2 LISFS

The Logical File [31]system is based on Boolean logic together with Logical Concept Analysis (a development of FCA where the result of logical formulae are used in place of binary attributes). Paths are formulae, directories represent queries and determine set of files and

parts of file whose description satisfies the formulae. The root directory represents the formula "true", and sub-directories of a directory are determined by the most general properties refining the query. The *ls* command gives hints about further queries that can be made. The file system is thus generated as a user explores it. Each directory is created by users' queries, the file system can thus be used for both navigation and querying. The names of the directories are taken from the queries used to form them.

From a file-system perspective the developers of LISFS consider arbitrary relations to be interpreted as a generalisation of symbolic links between two objects. Thus while in a regular UNIX system the symbolic link is only a reference to another file, in LisFS links between files form the basis for directory structure.

4.2.3 LISFS example: exploring two data structures

An example interaction taken from [85]:

```
[1]% cd /lfs/music/year:[1980..1990]
[2]% cd !genre:Comedy
[3]% cd time:<7min
[4]% cd .ext
[5]% playmp3 *
...
[6]% cd /lfs/music/genre:Disco/
[7]% ls
artist:BeeGees/ artist:DonnaSummer/
[...]
year:1976/ year:1977/
[...]
```

Shell Interaction 11 – *Command 1 selects music files from the '80s, command 2 excludes comedy files from the selected files and command 3 selects only short songs. For each one of the commands 1, 2 and 3 a different transducer (or attribute extractor) is used. From then, the user can switch to the extension of its query, which actually contains the files that match the query that has been built up (command 4) and listen to it (command 5). Command 6 selects disco music and command 7 then asks for possible navigation links. It can be seen that they can be used as possible queries.*

Some of what LISFS accomplishes could be done with appropriate use of the shell. However using LISFS is simpler and somewhat more intuitive. In addition all this navigate and query functionality can easily be used within applications (and also keeps those applications simple). Unlike libferris it uses standard UNIX utilities (all the intelligence is kept in the file system) but can only be run in GNU/Linux.

Like libferris, LISFS permits a diverse range of data to be used to build file-systems, using transducers to find appropriate attributes to use to build concept lattices. For example a file containing C source code can be mounted as a file-system.

Each of the lines in 12 is a different object in the context of the program file foo.c. Hence the concept table is :

```
[1]%cat -n foo.c
1 int f(int x) {
2 int y;
3 assert(x>1);
4 y=x;
5 fprintf(stderr, x=%d, x);
6 return y*2
7 }
8 int f2(int z) {
9 return z*4
10 }
```

Shell Interaction 12 – *The contents of a program source file.*

	function:f	function:f2	var:x	var:y	var:z	debugging	specification
1	x		x				
2	x			x			
3	x		x				x
4	x		x	x			
5	x		x			x	
6	x			x			
7	x						
8		x			x		
9		x			x		
10		x					

Now mounting the file allows navigation and querying:

```
[2]% mount foo.c / --transducers=c_transducer
[3]%cd /
[4]% ls
 debugging/
specification/
function:f/
function:f2/
var:x/
var:y/
var:z/
```

Shell Interaction 13 – *The source file is mounted as a file-system and can be browsed, the transducer is specified in the mount command.*

Changing directory to not debugging or specification - !(debugging|specification)) - gives a view of the file without those items.

Editing this file will still edit the main file. The logical file system is thus integrated with the shell, it permits the creation of a directory where there was not one before qualified by

```
[4]%cd function:f/
[5]%cd !(debugging|specification)
[6]%ls
foo.c var:x/ var:y/
[7]%cat foo.c
int f(int x){
int y;
............:1
y=x;
............:2
return y*2
}
............:3
```

Shell Interaction 14 – *A view of a source file that is restricted to those parts of the file that do not involve debugging code or specification code*

!(debugging|specification). There is always the option of getting information about the main global data set foo.c filtered via the navigated path changes. Boolean logic defines a set of base operations including conjunction here a search query is defined by the path navigated thus far and stored in a stack. The cd command thus acts to introduce an extra filter to the query. Navigating to '.. ' pops the filter query stack. Search results can be seen by doing ls. The file system and the shell navigation commands are thus an interface or a view to a more complicated data structure.

However this particular transducer maintains a subtle deficiency within traditional UNIX tools that treat input as arrays of lines. It should perhaps operate on higher level structures which are defined by regular expressions - so called structural regular expressions[86].

4.2.4 Review

In summary there are many commonalities between LISFS and ferrisfs they both use a plug-in or transducer system to convert data into file-systems, both can only be used with Linux, FCA (or LCA) is used in both as an aid to browsing search results. They also have sophisticated indexing systems to facilitate updating the concept lattice on the fly. There are differences in how Formal or Logical Concept Analysis is integrated into their respective frameworks, in libferris: FCA is not the default approach to representing data. Both are usable and demonstrate that the fundamental idea of using file-systems as a mechanism for representing data has considerable merit. The author of libferris in particular notes the power of integrating many heterogeneous resources into one common metaphor[70], this seems especially powerful with the applications he exposes to a file-system interface which in turn permits powerful extra functionality. They both also support a substantial and diverse number of data sets.

Both file-systems lack two features stemming from their Linux heritage: they are unportable and not easily networkable.

Other flaws with libferris with its implementation, can only fully be used by libferris

aware applications. This in turn has required rewrites of many common UNIX components. Since it provides an alternate file API, existing applications require changes in order to gain the full benefits of the libferris system. However the author of libferris has not then pursued the philosophy of using powerful yet small UNIX programs like `grep` `awk` or `wc` that have orthogonal purposes and leveraging the ability to chain them together. Instead for data transformation tasks libferris uses a XML Style Language Transformation(XSLT) processor by converting file-systems to XML representations then transforming then returning to a file-system.

LISFS transducers are structure and its logical operations can potentially replace many other system commands. This allows file system hierarchies to be created and categorised via logical operations.

There are various kinds of plug-in file systems that transform devices and data into file systems. It is an unsurprising lesson relearned by the developers of both libferris[70] and LISFS on the power of having a consistent file interface to dynamic structures.

In both cases many of the design decisions have chosen to utilise a different design philosophy then the one chosen by the creators of Inferno and Plan 9.

4.3 Design strategy

Before introducing a prototype system this section provides the motivations and decisions that underlie the system architecture. The architecture is guided by considering the general goals and workflow for exploratory data analysis and the UNIX and Plan 9 model of software engineering. The traditional UNIX philosophy is to build small tools, that do one thing well and that provide simple interfaces to other such tools. The interoperability between tools in UNIX is facilitated by almost all tools using a common interchange format of arrays of lines. Plan 9 develops this further by providing guidance on how to develop long running services and

With this philosophy in mind, the first step is to see if it is possible to decompose the required functionality of the system into simpler discrete components which can then be used together. This is done by first addressing what the key processes are within the system, examining the constituent data structures and looking at how putative components can be integrated to form a complete system. This section concludes with a short discussion on the general approach to the system architecture.

4.3.1 Key processes

Within this system there are four conceptually different processes necessary to form and work with a conceptually ordered data structure from a set of data:

1. Preconditioning, data transformation or feature extraction.

2. Concept formation or data mining.

3. Presentation and manipulation of the file system.

4. Manipulation and provision of ontological data.

Approaching these three processes in terms of their connection with each other, the first two processes of the system (transduction and concept analysis) are straightforward processes that take in input and generate output, they can straightforwardly be realized as programs. Processes 3 and 4 however are more appropriately realized as services.

4.3.2 Data structures

Formal concept analysis largely involves the manipulation of set based data structures. Sets can be efficiently represented within a computer by bitmaps, that is arrays of bytes.

Thus far in the thesis there have been two examples of partially ordered data-structure representing data in two different contexts, ontology and concept lattice. However within these two different contexts, there are two different types of usage and hence motivate two different representations within a file-system. With the Gene Ontology file system the usage is a straightforward query mechanism. If exploration of the Gene Ontology was the use scenario then it would make some sense to turn the Gene Ontology into a hierarchical namespace. Indeed an initial prototype was made with this scenario and style in mind.

The key motivation for the difference in adopted approach lies in the natural features of a file system, which are beneficial for a concept lattice FCA. These features would increase the complexity of implementation for the ontology without substantially increasing its usefulness or practicality. In particular it would result in awkward constructions (though not as awkward as the SQL seen in section 3.3.3) to carry out traversal operations.

In comparison the practical benefits of this system are that it can offer the theoretical framework and (an extensible form of the) semantics of relational databases whilst also providing the structure of object orientated data stores.

4.4 A prototype system

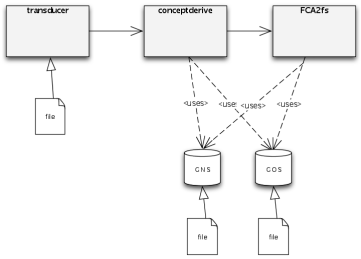


Figure 14 – *The relationship between the components of the system can be seen, the plain arrows show output piped between components whilst the the hollow headed arrows show data piped in from files. The dotted arrows show the*

4.4.1 Feature extraction - Transducers

The first stage in categorising files using this system is to generate a table of attributes, for this simple programs called transducers are used. They typically take a flat directory consisting of files to be categorised and create a binary table of attributes for those files. A transducer generates a table representation of the formal context described above in 2.2.1.1. The context table is simply a tab-delimited text file which has attributes as columns and objects as rows; and an x where every object has a attribute and a 0 otherwise. The format of this context table is designed to be both human readable and machine editable. Large context tables can be made simpler by either removing some attributes or by ORing some of them together. Selective filtering of the context table by standard UNIX or Inferno commands is straightforward, one can for instance use *grep* to select objects out of the context table or *awk* to select attributes.

Even for data that seems to complicated to use with just binary attributes for example in gene expression bi-clustering, with a suitable choice of transducer it is feasible to carry out coherent value bi-clustering. Inferno can use native programs on the host system using the OS command, and transducers can therefore be implemented in any programming language supported by the native operating system. However it is advantageous to develop transducers in Limbo since this guarantees portability. Transducers can pre-calculate context tables off line.

4.4.2 Concept formation - Conceptderive[3]

Formal Concept Analysis (FCA)2.2 is used to generate a categorisation of the data set. This takes a context table in the format described above in L2L2FCA and applies the standard FCA next closure algorithm to produce output listing the concepts (this is the objects and the

[3] A full description can be found in 2.3.

attributes that belong to that concept) and also lists the parents and children for each concept. The output is in the form of human readable s-expressions.

Optionally it also serves its own control and data file-system. This file system as seen in shell interactions 15 and 16 is based on the same design principles underlying many Plan 9 virtual file-systems such as the directory of processes or the directory of network connections examined in 4.1.5. This is a somewhat different approach data to representing concepts as directories compared to the libferris and lisfs file systems.

Whereas FCA2fs produces an unfolded lattice with concepts having multiple directories representing them. Conceptderive instead serves a list of numbered directories, with each concept corresponding to a single directory. The number of each concept in this directory directly corresponds to the qid number of the concept in the FCA2fs file system. The numbered Conceptderive file system is also a mechanism for manipulating concepts. Only three operations are permitted: an attribute can be removed, a concept can be added or an attribute can be moved. Why adopt a multiple file system method? This is to avoid some of the issues that both libFerris and LisFS run into: they need to modify or adapt existing tools in order to work best with their new file systems. Opening and writing to files in this second file-system is all that is needed to effect changes in the primary file system. This method is modelled on how Inferno and Plan 9 eliminate a plethora of system calls by using file accesses on generated files.

This file-system is always mounted as */n/concept.$pid* where *$pid* is the unique program id assigned by the system to the running instance of Conceptderive. The format of this file-system can be seen in 15

```
% ps | grep Conceptderive
9000     9000      abhey    0:00.0        alt    77K Conceptderive
% lc /n/concept.9000
Objects/
Concepts/
objhash
attrhash
conceptlattice
% ls /n/concept.9000/Objects
1
...
8
% cat /n/concept.9000/objhash
Leech Bream Frog Dog Spike Reed Bean Maize
% cat /n/concept.9000/attrhash
alive water land plants seeds2 seed1 animals legs mammals
```

Shell Interaction 15 – *This is the control file-system for conceptderive. The objhash and attrhash files give the names of the objects and attributes respectively.*

There are two principal pieces of functionality available from the file-system, it permits the adjustment of the attributes possessed by objects. These changes do not get passed to

the input file (this would make no sense if stdin had been used). Instead they can be used to generate a new concept lattice that is accessible via the conceptlattice file

```
% cd /n/concept.9000/Objects/1;ls
name
attributes
ctl
% cat name
Leech
% cat  attributes
alive water animals
% echo add 3 > ctl; cat  attributes
alive water land animals
% echo del 7 > ctl
alive water land
% cat /n/concept.9000/conceptlattice
(Length "19"
(Concept "0" (extent "0" "1" "2" "3" "4" "5" "6" "7" )
(intent "0" )(obj )(atr "0" )(sub "1" "2" "6" "11" )
(closure "0" "1" "2" "3" "4" "5" "6" "7" "8" "9"
"10" "11" "12" "13" "14" "15" "16" "17" "18" )
(super "0" )(subprime )(closureprime )
(superprime "1" "2" "3" "4" "5" "6" "7" "8" "9"
"10" "11" "12" "13" "14" "15" "16" "17" "18" )(terms ))
...
```

Shell Interaction 16 – *The file-system permits the manipulation of attributes of each object. Indexing starts from 1. Here object 1 (Leech) has attribute 3 (Land) added and attribute 7 (animals) removed. By reading conceptlattice file Conceptderive regenerates and outputs the modified concept lattice as an s-expression.*

4.4.3 Directory generation - FCA2fs

This takes the output from Conceptderive to generate and serve a synthetic file system (an Inferno server). Directories are formed from the concepts, however the concepts form a *lattice* which means that concepts can and do have multiple parents. To implement this poly-hierarchy every path through the lattice is traversed and for each path a directory is created. This means the order of names that make up a path from the root of the mounted file system is irrelevant and there are multiple equivalent directories, each of which point to the same contents and equivalent child directories. FCA2fs alters the namespace by mounting itself over the existing directory it is run in, however it spawns a thread that runs within the old namespace, access between the old and new namespaces are then connected together via a channel (see section A.3 on the threading and concurrency primitives).

4.4.4 File system traversal and naming

The two commands conceptderive and FCA2fs are typically run together with a pipe connecting them. FCA2fs can alter the namespace by mounting itself over the existing directory it is run in, while still running a thread with the old namespace. File requests and responses are connected between the two running threads such that all reads and writes to files get passed through directly to the original files. The net effect is that a flat directory gains structure. Additionally the compounds can be rerun on the new namespace. By spawning new namespaces for each new query this system produces a similar style of interactive querying and navigation as LISFS.

The main navigation operations to support for this system are increasing or decreasing specificity (corresponding to decreasing or increasing coverage). Increasing specificity is straightforward (change directory down to a sub directory). Broadening or generalising the reverse operation is not as well defined since there are multiple possible parents that could be destinations when generalising.

Each entry in the file system has a definite path (that which was used to get to that entry), this is used when "cd .. " is executed. However while each directory has only one parent as far as ".. " is concerned there are multiple concepts and parental directories that are actually available. The mechanism to resolve this is discussed in 4.6.

4.5 GO utilities

The Gene Ontology has multiple uses in this set of tools. The lookup of Gene Ontology terms from gene names is done via a service called the Gene Naming Service (GNS) and is modelled on how the DNS service is implemented in Inferno as explained in section 4.1.5.2. For this service a file called /info/GNS is present in the namespace. In an analogous manner to the DNS service, the file is opened and a file handle obtained then a request is written to the file, and the result is available upon reading from the same file handle. It currently supports three annotations (or mappings), one based on locus ID, one based on Affymetrix's annotation files and one based on HUGO[14]. In all cases it converts the gene or probe symbols to GO terms.

Access to GO terms and their structure (e.g. finding hypernyms, calculating significance scores) is implemented in the same manner, as a file upon which read and writes carry out queries or calculations.

4.5.0.1 GOS reloadable implementation

The implementation of this module features a reloadable structure such that the main algorithms used to process the data can be rewritten and recompiled whilst the ontology data that has been parsed in or calculated from a file remains in the memory of the still running

```
% ls /info
/info/GNS
/info/go
% whatis hgid
fn hgid {  q=$*;
<>/info/GNS {
    echo -n hugo $q >[1=0];
    cat
    }
}
% whatis g
fn g {q=$*;
<>/info/go {
    echo -n $q >[1=0];cat}
}
% hgid DRD4
6021 16020 16020 7268 7212 7212 7186 7165 5887 5886
4952 4952 4952 4872 1584
% for ( i in `{hgid DRD4 }) { g cd $i;g pwd} | uniq
6021 myo-inositol_biosynthesis
16020 membrane
7268 synaptic_transmission
7212 dopamine_receptor_signaling_pathway
7186 G-protein_coupled_receptor_protein_signaling_pathway
7165 signal_transduction
5887 integral_to_plasma_membrane
5886 plasma_membrane
4952 dopamine_receptor_activity
4872 receptor_activity
1584 rhodopsin-like_receptor_activity
%
```

Shell Interaction 17 – *An interaction showing how the GNS and GOS service are accesed. The shell function hgid converts hugo gene IDs to GO terms. In this function the file /info/GNS is opened and the string 'hugo $q' is written to it (where $q is a variable that is passed in as the rest of the arguments to the function hgid) then the same file handle is read to obtain a reply. The ontology annotations for the gene DRD4 are extracted as numbers. The shell function g together with the two GOS commands 'cd' and 'pwd' are then used to obtain the name of the terms.*

program. Thus from the running program, it is possible to experiment with alternate scoring methods. This idiom was found to be particularly useful in the development of this module, since parsing data in from the Gene Ontology file took up the bulk of the run-time of the Gene Ontology program. This provides somewhat similar functionality to the "Fix and Continue" feature found in the XCode[4] development environment. This is also a common idiom in interpreted languages such as TCL and bash shell scripts.

This is implemented by having two modules, the GOloader and the the GOS module. The ontology file is parsed upon initial start up, and the contents of the ontology is stored in heap memory. The GOloader program spawns a process which in turn calls upon the GOS module. This process is accessible via a file in the file system - /info/go. Any writes to the file are channelled by the kernel to the GOS module. The GOS module is a file server. GOS accepts command requests and the replies are available from reads of the file /info/go. In carrying out the commands the GOS module calls the run function in GO if the command is not 'l'. In the case of 'l' being sent, the GOS module instead reloads the GO module.

This is a form of code reuse, where the limbo programming language and its compiler are used to define my interactions with my code. This style of programming has a LISP-like flavour to it. Though it is far from a perfect solution: to form a new query one has to edit source code and recompile.

4.5.0.2 Distributed middle ware

Both the GOS and the GNS components do not need to run on the same computer as the remainder of the Concept based File-system, the services are trivially exportable via the `styxlisten` command. This allows benefits such as better performance, by having each component run on their own instance of Inferno on another machine.

There are many methods of producing distributed middle ware including Corba, RMI or SOAP web services. These services could have been implemented as any one of these. Web services would for instance have the benefits of possibly offering some validation of input data. However none of them is quite as easy to implement within Inferno and the semantics of component access via file access (when using the Styx based remote file system protocol) is simple, familiar and extremely effective from a client perspective. Implementing a service as a file-system server is somewhat more difficult then say implementing a web service, however this is more due to a lack of tutorials and examples then any intrinsic difficulty.

In the current implementation whilst the GNS is usable by multiple users and processes simultaneously the GOS is not tested, expected or guaranteed to be so. This is entirely due to the stateful functionality of being able to edit and reload the key query code of the module. This is not insurmountable and it would be straightforward to re-architect the service such that every process or user attaching to the GOS would spawn their own dedicated process and their own particular code could be loaded for their session.

[4]Xcode is a code development environment found on OS X which permits developers to write and compile code.

```
% whatis gnsspawn
fn gnsspawn '{os emu ''-pheap=120000000'' -c1 /dis/sh.dis -c ''{
    ndb/cs
    load std
    /usr/abhey/dis/gns
    styxlisten tcp!*!7001 export ''''#sGNS''''    }''&
}'
% whatis gospawn
fn gospawn '{os emu -c1 /dis/sh.dis -c ''{
    ndb/cs
    load std
    mount -a tcp!127.0.0.1!7001 /info
    /usr/abhey/dis/GOloader
    styxlisten tcp!*!6889 export ''''#sgo''''     }''&
}'
% mount tcp!127.0.0.1!6889 /info
% mount tcp!127.0.0.1!7001 /info
```

Shell Interaction 18 – *Both gns and GOloader as services are accessible from the #s device*

4.6 Implementation of poly-hierarchy handling

The architecture for file systems within Inferno is arranged so that all system calls on files on are translated into requests and responses that are handled by a server, where a server can be a user program such as FCA2fs. Each instance of a file or a directory within the synthetic file system has a unique 64 bit unsigned integer allocated to it. This integer is used by the file server to identify requests for that file and is referred to as a QID.

When FCA2fs turns the lattice structure into a tree, the concepts are enumerated and each directory (and its contents) corresponding to a particular concept has the concept number encoded as part of the top 20 bytes of the 64 bit QID.

Thus while each instance of a file and a directory has a unique QID, equivalent files and directories can be identified. For example `Qidc` is a gate for the fs[1] system in Inferno that takes an integer, it is open if the entry has a QID that contains the given concept number, so the command shown in interaction 19 returns all directories that are part of concept 5.

```
% fs select {and {mode +d} {qidc 5} } $conceptserver
```

Shell Interaction 19 – *Finds all entries in the FCA file system that have QID 5*

In order to get to a more general upward (referred here in shell interaction 20 as a cddd) command the following procedure is used:

1. The qid of the current concept directory can be found by using the `ls -q` . In particular the first 20 bits of the 64 bit qid are the same for all directories and their contents that arise from the same concept.

2. Then the paths of the file server can be traversed via the fs shell command language. A gate (or filter) `qidc` with the qid of the current concept is used upon each traversed directory. `Qidc` filters by just the first 20 bits hence it is straightforward to lists all paths of a concept. They are all listed with their full path from the root of the synthesised file system.

3. From this list an entry can be chosen for navigation.

This system is not perfect, cddd only lists possible parent directories rather then offering a choice and allowing the selection to be made. There are some situations where ".." should be used and others where cddd is necessary and arguably the user should not be required to know there is a difference or have to behave differently.

```
fn cddd {
qid=`{echo `{ls -q | tail 1 | sed 's/\(/0x'} & 1<<19|mathcalc}
fs select qidc $qid $fileserver
}
```

Shell Interaction 20 – *Shell script that lists parents that can be navigated to*

4.6.1 Using the file-system in the native environment

On both Linux and OS X it is possible via the Plan9Ports[5] project and the File system in User Space[6] to mount the hosted Inferno file-system as a native file-system. To do this Plan9Ports must be installed in addition to Inferno, the file-system in Inferno needs to be exported as in shell interaction 21.

```
%
% mkdir /tmp/pipe
% bind '#|' /tmp/pipe
% <>/tmp/pipe/data{export /n/FCAfilesystem &}
% os 9pserve inferno < /tmp/pipe/data1 >/tmp/pipe/data1
```

Shell Interaction 21 – *Commands to be run from within Inferno.*

and the file-system needs to be mounted in the host environment

```
~/$ 9 mount  `namespace`/inferno /mnt/inferno
~/$ cd /mnt/inferno
~/$ ls
```

Shell Interaction 22 – *Commands that are run on the native (OS X) host*

[5] http://swtch.com/plan9port/
[6] http://fuse.sourceforge.net/

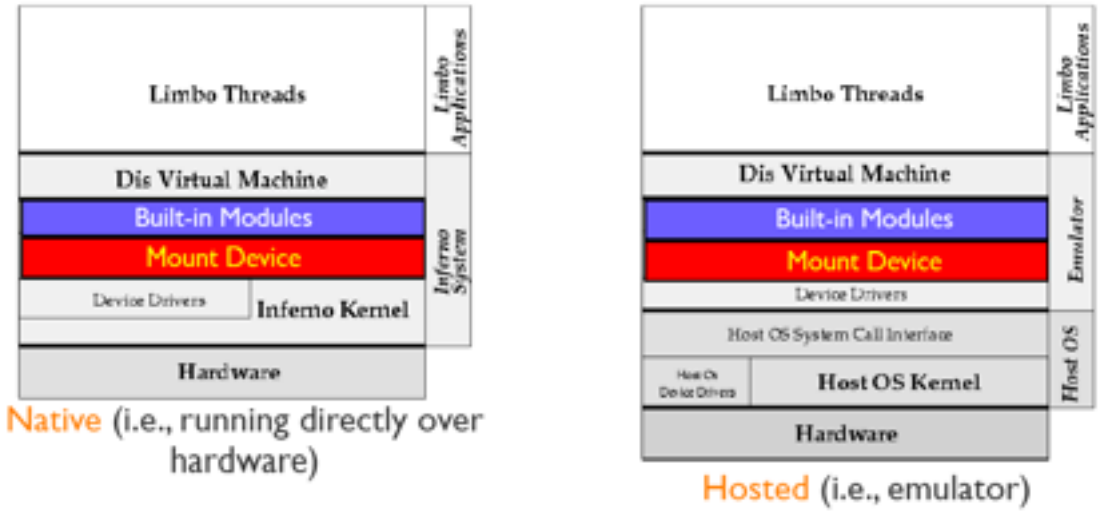
4.7 Discussion

The architecture of this concept based file system has the same goals and features as the other two main FCA based file systems libferris[70] and LISFS[85] however its implementation is very different. Principal changes are based on using two separate file systems and a set of simple shell utilities.

Whilst this system does leverage the unique facilities available within Inferno almost all the power is still available when the system is mounted under a different operating system. Some of the key benefits of this architecture are that much of the difficult or ambiguous code can be kept out of the file system. It also offers a solution to implementing deletes and moves within the file system, disallows them unless they are carried out from the control file system where the semantics can be altered to reflect the actual data structure being modified. It benefits from the superior portability of the Inferno system, it is the only one that can be demonstrated (in hosted form) under the Windows Operating System.

There are numerous improvements that could be made, a somewhat obvious one is to make the concept derivation (the calculation of the closure operator) that occurs as the user traverses through the file system restricted to only one level. This has two benefits firstly this dynamic file system would amortise the calculation of the concept lattice over the time whilst the user is using it. Secondly it would allow the transducer to be be modified and in turn altering the relation.

Figure 13 – *The architecture of the Inferno environment native and hosted (image from[108])*



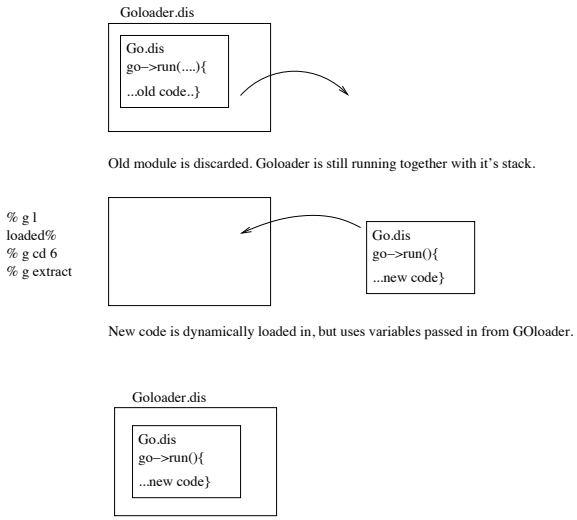


Figure 15 – *This picture demonstrates how code is dynamically loaded into a running process.*

Chapter 5

File-system based concept exploration and navigation

This chapter demonstrates the knowledge management system, ConceptOntoFs, operating on a set of real data sets. This includes an introduction and characterisation of the principal data set that is considered: results from human gene expression micro array experiments.

This particular data set is an example of a typical post-genomic data analysis challenge: to find structure and patterns in multivariate data sets representing the concerted behaviour of multiple genes in response to different treatments/perturbations. A variety of pattern-based clustering techniques have been developed and experimented with to extract meaningful clusters from such gene expression data [101, 24, 35, 103]. These methods can help in multiple applications including gene classification, identifying unknown functions of a gene or providing insight about the mechanism of gene regulatory systems[97]. As explained in section 2.1.2 a special type of clustering algorithms, bi-clustering algorithms (FCA is a form of bi-clustering) - have shown efficacy at identifying meaningful patterns in gene expression data sets [67]. In order to best uncover the patterns within the numerical multivariate gene expression data, the data needs to be transformed into a format suitable for applying the Formal Concept Analysis algorithm. This chapter describes an order preserving technique to perform this transformation and shows the results of applying this method to a gene expression data set. A typical use case that is explored for these methods is the desire for a biologist to learn about genes which have a similar behaviour to a gene of interest and the experimental conditions under which these similarities occur.

The system is also demonstrated structuring and aiding in the navigation of a data set representing a digital music catalogue, this illustrates the modularity and generic capabilities of this system.

I also address the three principal issues that are raised when trying to use the system with large data sets (of the order of 11000 by 500), these include the temporal performance of the

clustering algorithm, memory requirements and the final usability.

The Chapter concludes with a discussion of the characteristics and performance of the system.

5.1 Data sets

5.1.1 Miscellaneous data sets

5.1.1.1 ID3 tags

A quick and simple usage of the system is to use it to arrange a number of music files with ID3[1] tags.

5.1.1.2 The mushroom data set

A popular data set that has been used for many data mining studies (including to assess the speed of Formal Concept Analysis algorithms and implementations) is the mushroom data set found in the UCI Data Repository. This data set consists of 8,124 records describing various kinds of mushrooms as found in the The Audubon Society Field Guide to North American Mushrooms [61]. [2]

The data is contained within one file - agaricus-lepiota.names, it is formatted with one line devoted to each mushroom. Each mushroom is described by 22 nominal attributes - each one of which can take a number of different values - and results in over 100 binary attributes.

5.1.2 Gene expression data

5.1.2.1 General considerations

As explained in section 2.1.2 genes provide a key role in the ongoing physiological processes that occur within a living cell. The effect of various stimuli upon the cell cause proteins binding upon a binder site of a gene sequence, this in turn causes the production of a string of RNA which in turn provides a set of instructions for the ribosome to generate a particular protein. These proteins lead to phenotypic effects including regulating the expression of other genes. DNA micro arrays[101] in turn permit the measurement of thousands of genes expression levels simultaneously. This technology thus easily allows detecting the context-dependent differential change in expression levels.

Moreover in order to identify genes that are co-regulated and co-expressed in a subset of conditions but behave in an independent manner in other conditions sophisticated techniques such as bi-clustering[67] are employed.

[1] ID3 is a popular music tagging format that allows attribute value pairs to be embedded within music files. More details can be found here http://id3.org/.

[2] The data can be downloaded from http://mlearn.ics.uci.edu/databases/mushroom/

5.1.2.2 Interesting genes

The L2L data-set[78] is curated by hand directly from the gene expression data of a diverse subset of publicly available human gene expression experiments. This is the simplest data set to start analysis from, it simply consists of a set of files each containing a list of genes that were significantly up or down regulated in each experimental study. Where the details are available the l2l data-set also records whether the genes of interest were up or down regulated however no record is kept of the magnitude of over or under regulation. This is done since fold-changes are often incomparable across different platforms

Each experimental condition is stored within a separate file, each file has a small header and then contains a list of the genes - referred to using the HUGO naming convention - that were interesting in that study. The objects O are the studies (the files) , the attributes A are the genes and the relation I between is defined by oIa if gene a is in the file for study o. Each of the genes has a Gene Ontology Annotation so Gene Ontology terms are available to label the concepts that are found. The L2L project consists of 595 lists, altogether covering just under 11 000 genes of interest.

5.1.2.3 Array Express

A principal source of gene expression data is the Array Express [3] database. This public resource contains over 2000 gene expression experiments which together represent over 60000 samples (or experimental conditions).

Each experiment provides for each experimental condition and for each gene probe whether a transcript was present, marginal or absent. In this context absent means the expression level is below the threshold of detection of the micro array and therefore indistinguishable from zero. Whereas marginal is close to the threshold of detection. Also provided is a processed signal value of the abundance of a gene. This signal value has typically already been pre-processed by treating experimental artifacts such as outliers, missing data and averaging over array replicates.

For each experiment this processed data is provided in multiple formats including a plain text tab delimited file format, with the values of one probe (and an associated p-value) under the various experimental conditions to a line . In order to use this numerical data it is neccessary to extract just the signal value required and to transform it into a form suitable for formal concept analysis use. This is discussed later in this chapter.

[3] Available from http://www.ebi.ac.uk/arrayexpress/

5.2 Computer Experiments

5.2.1 Music files catalogued using ID3 tags

In order to produce attributes for an mp3 music file a pair of tools (written in limbo) named `id3` and `id3parse` are used. The command `id3` takes an mp3 file as input and extracts the fields in the id3 tags associated with it. By default it then prints the genre and album fields with tabs between the two fields. To create a context table `id3parse` takes as input a file name then the output of `id3` and uses a hash table to build the binary matrix representing the mp3 files. This can then be sent to `conceptderive` and `FCA2fs`.

With the `p` option to FCA2fs the file contents are taken from the directory where FCA2fs is started. In the shell interaction 23 FCA2fs takes an unsorted list of music files and sorts the files into directories according to their genre and their album. The full directory structure can be seen in figure 15

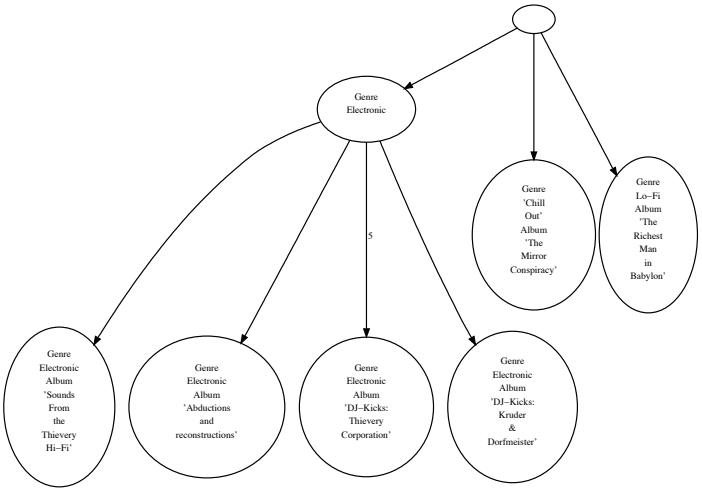


Figure 16 – *Derived directory structure of mp3 directory.*

5.2.2 Benchmarking

There are several algorithms for computing computing concept lattices, the most efficient for general practical applications is commonly accepted to be Next Closure [109, 33]. The Audubon Society Field Guide to North American Mushrooms [61] data set is frequently used to benchmark the performance of FCA implementations [3]. The `parsemushroom` command takes every line of the raw data set and constructs a binary matrix from it representing each object by the binary vector of 22 attributes.

The implementation of the Next Closure algorithm in Limbo is far slower then a native C implementation. This is primarily the naivety of the implementation, which despite efforts to

```
% pwd
/usr/abhey/music
% ls
01_David_Byrne___Dance_on_vaseline.mp3
01_Tropicando.mp3
10_Shaolin_Satellite.mp3
09_Coming_From_the_Top.mp3
18_It_Takes_a_Thief_DJ_Kicks.mp3
01_A_Warning_(Dub)_(feat._Hutchy).mp3

...
% cat 01_Tropicando.mp3 | id3
Genre Electronic        Album 'DJ-Kicks: Thievery Corporation'
% ls \
| getlines {echo -n $line^'  '; cat $line|id3} | parseid3 \
| conceptderive | FCA2fs -p
% ls
Genre 'Chill Out'Album 'The Mirror Conspiracy'
Genre Electronic
Genre Lo-FiAlbum 'The Richest Man in Babylon'
% cd 'Genre Electronic'
% ls
01_A_Warning_(Dub)_(feat._Hutchy).mp3
01_David_Byrne___Dance_on_vaseline.mp3
01_Tropicando.mp3
02_2001_Spliff_Odyssey.mp3
02_Rebirth.mp3
03_Beija_Flor.mp3
...
Album 'Abductions and reconstructions'
Album 'DJ-Kicks: Kruder & Dorfmeister'
Album 'DJ-Kicks: Thievery Corporation'
Album 'Sounds From the Thievery Hi-Fi'
% cd 'Album ''DJ-Kicks: Thievery Corporation''';ls
01_Tropicando.mp3
02_Rebirth.mp3
03_Beija_Flor.mp3
04_Mother_Africa_Feeding_Sista_India_-_2001.mp3
05_Rainbow.mp3
06_Success_(Thievery_Corporation_remix).mp3
07_Emerald_Alley.mp3 08_Exploration.mp3
...
cat 01_Tropicando.mp3 | os mpg123 -
High Performance MPEG 1.0/2.0/2.5 Audio Player for Layers 1, 2 and 3
  version 1.3.0; written and copyright by Michael Hipp and others
  free software (LGPL/GPL) without any warranty but with best wishes
Playing MPEG stream 1 of 1: - ...
Title:   Tropicando
Artist: Les Baxter Album:
DJ-Kicks: Thievery Corporation
Genre:   Electronic,
MPEG 1.0 layer III, 192 kbit/s, 44100 Hz joint-stereo
```

Shell Interaction 23 – *The command extracts the ID3 tags to form attributes which are in turn used to produce a structured file-system. The p option to FCA2fs tells it to mount the new file-system over the present directory, and reroute all file reads to the original file. Thus the presented file-system can be used to read from. In this case it is played through an mp3 player(mpg123) on the host system.*

profile and improve its performance does not scale particularly well to massive data sets.

Some speed increases were gained from storing arrays of binary data using arrays of 64 bit unsigned integer (referred to as a *big* in Limbo) rather then a byte and by optimizing when the garbage collector is called. The implementation would probably receive a significant speed increase if it used a D-Gap based compression scheme to store bit vectors [4] .

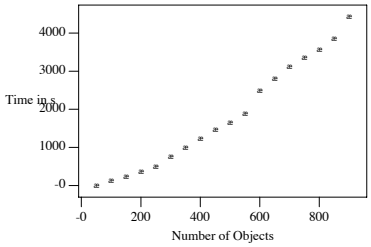


Figure 17 – *A graph of run time against number of objects.*

The graph illustrates the run time performance of constructing a lattice from subsets of the Mushroom data set using a dual processor Athlon 1400Mhz with 2Gb of Ram running Inferno upon Plan 9. For large concept lattices memory issues and the computational cost of calculations overwhelms the system's ability to cope.

While it is still considerably slower then the Next Closure implementation in C, it is quick enough to be useful for small concepts - which in general are more usable then large concept lattices for key conceptual knowledge processing tasks[33]. This practical limit in part forced me to develop transduction techniques to create small useful concept lattices.

5.3 Biological Experiments

5.3.1 L2L data set

One approach to exploring the L2L data-set would be to place the entire data-set into one context table and then derive a concept hierarchy from that. However within the L2L data-set there are relatively few genes in common between the experiments. Whilst there are 595 experiments, the average number of experiments any gene appears in is less then 5[5]. Taking all experiments together to form a context matrix would produce an extremely sparse matrix. This in turn would result in a flat concept hierarchy, with each concept cluster consisting of just one experiment. Flat concept hierarchies and in turn a flat concept file systems do not tend to increase the explorability of data sets or provide the user with novel inferences. Though from a pattern mining perspective, a flat hierarchy still represents a set of disjoint and

[4] An efficient bitset library based on D-Gap coding for C++ can be found here http://bmagic.sourceforge.net/index.html.

[5] To calculate an upper bound the follwing command may be used: *cat * | sort | awk '{a[$1]++} END{for (i in a) { s+=a[i];t++}; print s/t}'* .

easily distinguished clusters: i.e. still a potentially useful clustering. There are performance issues related to mining the L2L dataset, with a large number of sparse concepts it was slow to run but before it could complete the memory requirements exhausted the system. Thus it proved impossible to run the system with the entire L2L data-set.

With the L2L data set a simple program written in limbo called `l2l2fca` takes one or more list files from the L2L projects and uses a hash table to generate a binary matrix suitable for passing to conceptderive. The L2L data set one can instead search through the experiments and only include those that involve a gene (or genes) of interest. Thus to identify the experiments and genes that share some commonality with the BRAC1 gene, a gene associated with breast cancer, the command in shell interaction 24 generates a binary matrix associated with the BRAC1 gene.

```
%cat `{grep -l '^MALT1$' *} | l2l2fca
```

Shell Interaction 24 – *Generating a binary matrix that structures all experiments involving the MALT1 gene.*

The command in shell interaction 25 can be used to examine all experiments that share a gene in common with an experiment of interest.

```
% x=`{cat werner_fibro_up | grep -v '^#' }
% searchstring='('^`{echo $x | sed 's/ /)|(/g'}^')'
% cat `{grep -l $searchstring *} | l2l2fca
```

Shell Interaction 25 – *Finding all experiments that share genes with an experiment. The variable x contains the list of genes in the experiment of interest*

By using the technique described in shell interaction 24 and focusing on a single gene of interest, the set of concepts in table 13 are found. Note that the terms that are derived are taken from the conceptderive part of the labeller algorithm, thus the bottom concept (concept 0) which contains only the gene MALT1 has GO terms that are below the threshold of the lattice. That is the derived concept lattice has depth 5 and MALT1 is only annotated to the general terms shown in table 14 all of which have a scaled depth greater then 5 in the ontology.

It can find putative terms appropriate for each concept node. These are annotated next to each concept node.

A reduced form of the concept lattice with the edges named is shown in figure, in the real system the lattice is unfolded the lattice to form the file-system.

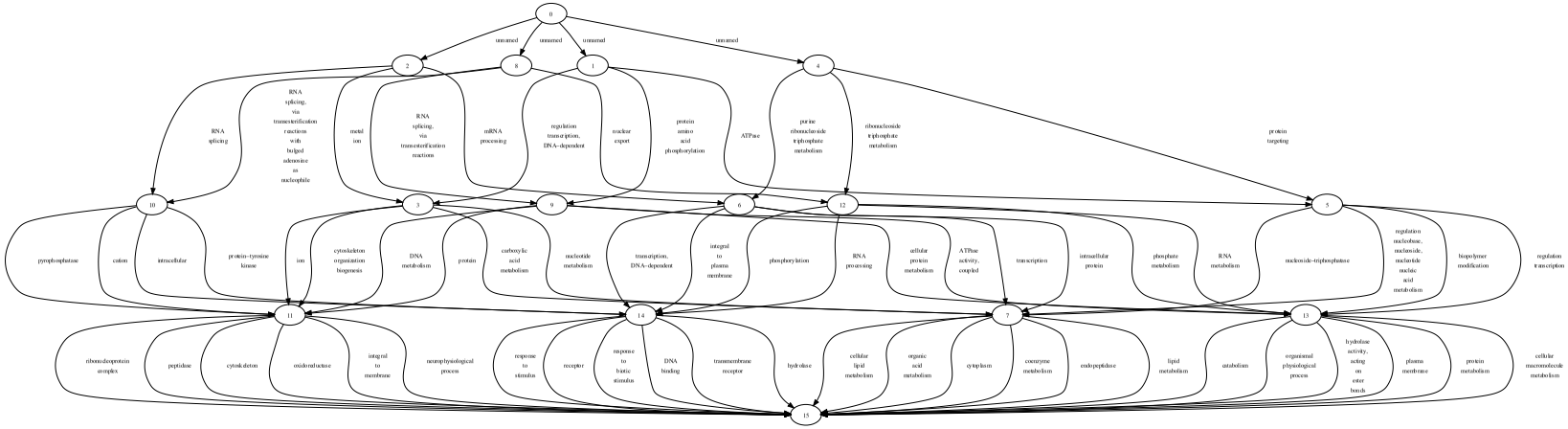


Figure 18 – *Reduced concept lattice of the MALT1 gene. Only half the genes are used in this representation.*

5.3.2 Order preserving bi-clustering

Only considering the presence or absence of a particular gene in an experiment is a somewhat crude method to examine gene expression data, there is considerable raw signal information about the abundance of a particular gene available from gene expression arrays. In order to do this effectively I want to describe an alternative approach to the simple general scaling methods for transforming numerical data as presented in section 2.5. The use of this method is motivated by the particular *biological* reasons which underlie why particular gene expression data should be clustered together. A typical motivating goal to analyse gene expression profiles is in order to understand the different regulatory interactions between genes[35, 4].

Ben-Dor et al. [12] introduced the technique of OPSM (order preserving sub matrix) to discover a subset of genes identically ordered among a subset of conditions. With this method the emphasis is upon the detection of similar relative order of the conditions thus identifying the coherence of the actual expression values. A variant of this coherency approach is modified and implemented here. its implementation takes the form of a novel (to the FCA discipline) method of scaling numerical gene expression data into set membership data, the basic theory of which is described here.

Starting from the initial data set $\mathcal{D}$ an m by n real valued matrix containing the results of m experimental conditions (arranged in columns) each of which shows the resultant expression of n genes. Each row of the matrix is therefore a numerical vector that corresponds to the behaviour of a single gene, giving the expression abundance of the gene in each of the m experimental conditions. For each gene the corresponding expression vector is numerically sorted and from this a a ranking or ordering of the experimental conditions is generated. Furthermore in order to account for the noisiness of gene expression data, an equivalency relation can be used to group experimental conditions that have approximately the same level of gene expression.

Concepts	Objects	Attributes	Terms
0	uvc_xpcs_8hr_dn uvc_xpcs_4hr_dn uvc_ttd-xpcs_common_dn alzheimers_incipient_up alzheimers_disease_up	MALT1	nil
1	uvc_xpcs_8hr_dn uvc_xpcs_4hr_dn uvc_ttd-xpcs_common_dn alzheimers_disease_up	MALT1 C19ORF7 NFIB CDC2L5 IGF1R E2F5 KLF7 WRN C4ORF8 CENTB2 TIPARP PPFIA1 MARK3	6375 244 387 16540 6739 398 377 375 42623
2	uvc_xpcs_8hr_dn uvc_xpcs_4hr_dn uvc_ttd-xpcs_common_dn	MALT1 C19ORF7 NFIB CDC2L5 IGF1R E2F5 KLF7 WRN C4ORF8 CENTB2 TIPARP PPFIA1 MARK3 APPBP2 TRIM33 ZFHX1B PHLPPL UBR2 KLF6 LARP5 RASA1 RASSF3 BDNF DOC1 DOCK4 RUNX1 SLC25A12 HEG1 TRIM2 CDK8 VPS13B TIAM1 EDD1 NAV3 ACSL3 ABCC1 MTAP GLRB IRS1 HOXB2 ASXL1 HRB TSC22D2 MICAL3	6375 244 387 15822 5237 45767 6020 16933 16540 30041 16408 6739 19319 46364 8064 43066 30832 15807 6839 8154 82 398 377 375 6769 51258 8380 4714 7249 7169 5231 51028 19362 19199 6397 16485 6733 42113 6916 30029 30098 16071 50658 50657 6396 6405 30036 19318 30005 51168 6875 45934 15630 15698 16051 6865 6820 19941 4713 46942 45449 51169 6351 44257 51603 7067 6366 16310 46649 45321 30163 6913 6464 6357 17111 42623 6468 16887 6508 6355
3	uvc_xpcs_8hr_dn uvc_xpcs_4hr_dn alzheimers_incipient_up alzheimers_disease_up	MALT1 NCOA3 RBPSUH FYN SRPK2 QKI C13ORF24 ZNF198 ZHX3 RNGTT MYO10 CAP350	4468 398 377 375

Table 12 – *The concepts from the MALT1 gene. The terms are sorted by scaled coverage/informational value.*

GO ID	Definition
42981	regulation_of_apoptosis
30693	caspase_activity
8233	peptidase_activity
7250	activation_of_NF-kappaB-inducing_kinase
6952	defense_response
6916	anti-apoptosis
6508	proteolysis
5622	intracellular
5515	protein_binding
4871	signal_transducer_activity

Table 13 – *GO Terms for MALT1*

	Experimental Conditions							
	1	2	3	4	5	6	7	8
Genes								
A	318.2	134.6	180.7	332.9	205.7	153.8	84	81.5
B	251	126.4	166.4	184.5	161.9	96.6	147.9	166.4
C	890	270.7	286.3	455.9	352.1	214.4	160.6	241.6
D	716.8	343.9	348.9	553.7	496.8	258.2	283.1	250.6
E	775.1	417.7	533.3	731.7	596.9	378.7	354.6	387.9
F	4584.4	2249.7	2602.6	3226.9	3147	2015.6	2075	2173.1
G	766	436.6	315.9	662.2	462.9	231.9	252.3	333.8
H	12602.4	5172.2	6278.3	8869	7316.3	5403.2	5698.7	5484.4
I	8940.8	3698.5	4332.6	4615.7	4621.6	3316.5	3516.3	3343.2
J	2.5	1.6	3.9	4.9	2.4	0.8	12.2	6
K	18	7.3	9.3	1.9	5.3	7	22.4	24.6

Table 14 – *Showing a set of gene expression values under 8 experimental conditions.*

5.3.2.1 Order equivalency relation

Let A be the set of experimental conditions (column labels) and O the set of genes then define a function $D : O \times A \rightarrow \mathbb{R}$ that returns the expression value of a given gene under a specific condition. Then for a given gene $o \in O$ and a grouping threshold δ, where typically $0 < \delta < 1$, $A' \subseteq A$ forms an order equivalent subset if

$$\max_{a_i, a_j \in A'} |D(o, a_i) - D(o, a_j)| < \delta \times \min_{a_k \in A'} D(o, a_k)$$

and

$$\forall a_i \in A', \min_{a_j \in A'} |D(o, a_i) - D(o, a_j)| < \min_{a_k \in (A \setminus A')} |D(o, a_i) - D(o, a_k)|$$

The grouping threshold δ changes whether the expression of a gene in two different conditions

is considered equivalent, large δ will result in a fewer yet larger order equivalency subsets. A template can then be formed of a gene by the sequence of ordered equivalency subsets. Each gene's behaviour can be described in two ways: the ordering of expression values and by a template.

Thus every gene is represented by a template of an ordered sequence (or permutation) of experimental conditions.

5.3.2.2 Genes as queries

Picking one gene from the data set gene q, it can be described by the ordering of expression values. If this ordering of values is compared to another genes sequence of ordered equivalency subsets under the same conditions. Then whenever a gene behaves somewhat similarly to gene q it will match some part of the sequence of ordering of conditions. The subsets of the ordering that are matched gives the experimental conditions under which this gene behaved coherently with the query gene. From this matching process a binary vector is produced for each gene.

Carrying out the same procedure over every in turn gene, forms a binary matrix, which is used as a formal context to create a concept lattice. The binary matrix has each gene as a row with each experiment (or experimental profile) as a column. After transposing the binary matrix it can be used with conceptderive and FCA2fs.

The template for each gene is stored using a simple array, a more efficient data structure could be used such as in OPC[64] where the row IDs and the order of the columns are stored in a suffix tree construction, however the performance of a simple array is more then adequate for typical access.

With this formal context, the concept lattice has the following features: the original gene is at the bottom and similar genes are arranged such that the ordering underlying the concept lattice respects the coherency of the genes.

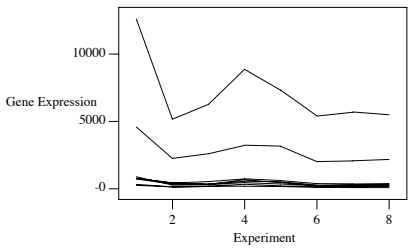


Figure 19 – *The gene expression of 8 different genes under 10 different conditions. There is a large difference in the ranges that different genes can take.*

For example in table 5.3.2 the gene expression of five genes under 10 different experi-

mental conditions is shown.

The first step of the process is to represent each row (therefore each gene) as the permutation of the column labels under which the gene expression values are increasing, then create a template by treating experimental conditions that are similar as equivalent. Hence for gene A, looking at the first row of table 5.3.2 the experimental conditions in increasing order are then 8,7,2,6,3,5,1 and 4. With a threshold δ of 0.2 then this forms four order equivalency sets: (8,7), (2,6),(3,5),(1,4). Comparing with gene C - which has the experimental conditions in the following order: 7,6,8,2,3,5,4 and 1 - the following context vector is generated (X,0,0,X,X,X,X,X,X). An X signifies that the queried gene (C) experimental condition matches an entry in the equivalency sets of the query gene (A). Repeating for every gene - with the additional constraint that if there are fewer then 50% (4 in this case) Xs then the row is deleted - yields a context matrix for gene A (table 5.3.2.2).

	Experimental Conditions							
Genes	1	2	3	4	5	6	7	8
A	x	x	x	x	x	x	x	x
C	x	0	0	x	x	x	x	x
D	x	0	0	x	x	x	x	x
E	x	0	0	x	x	x	x	x
F	0	x	0	x	x	x	x	x

Table 15 – *The context table generated by looking at genes similar to gene A.*

This context matrix in turn can be used to form a concept lattice as in figure 5.3.2.2.

The procedure can be repeated with any of the other genes from the original selection, table 5.3.2.2and figure 20 illustrate what happens when gene B is focused upon. Gene B's behaviour is very different from gene A. Hence neither appear upon each others concept lattice.

	Experimental Conditions							
Genes	1	2	3	4	5	6	7	8
B	x	x	x	x	x	x	x	x
C	0	0	x	0	x	x	x	x
D	0	0	x	0	x	x	x	x
E	0	0	x	0	x	x	x	x
F	x	0	x	0	x	x	x	x
G	x	0	x	x	0	x	x	x
H	0	0	x	x	x	x	x	x

Table 16 – *The context matrix of gene B.*

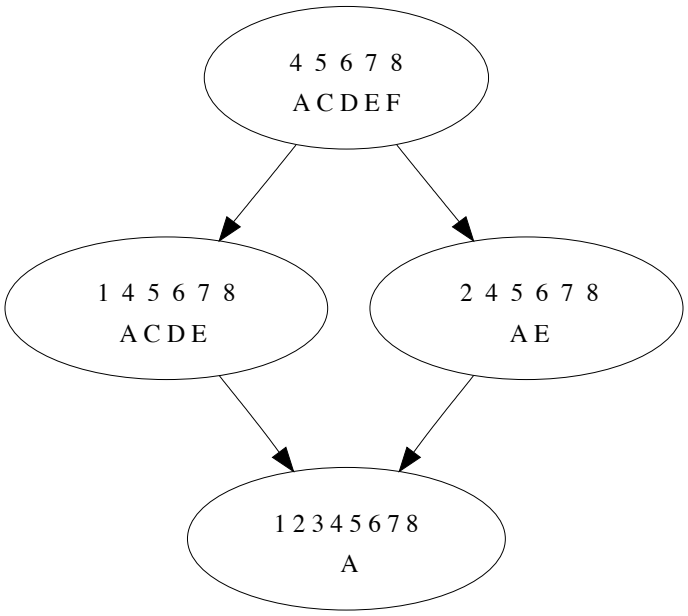


Figure 20 – *Concept lattice of genes similar to gene A. The experimental conditions are in the top line of the concept nodes and the genes in the bottom line.*

5.3.2.3 Arrayexpress2FCAns

The implementation of the above order preserving algorithm is via a separate program - Arrayexpress2FCAns. This component is again represented as a file service, upon starting the program it scans through the source data set building a representation of every gene, it then mounts itself onto `/info/ARREXP`. Upon writing a gene name to the file `/info/ARREXP`, a subsequent read using the same file handle will return a binary context table.[6] This context table as before is sent to conceptderive and FCA2fs.

Once the Arrayexpress2FCAns service is loaded with genes and their expression values. The Arrayexpress2FCAns service, can also provide subsets of the original raw numerical data if it is called by prefixing c to a list of genes of interest, it then returns the numerical values of those genes under all experimental conditions.

DRD4 is annotated to the terms in the Gene Ontology listed in table 5.3.2.3. Taking the gene probe for DRD4: 208215_x_at and forming the context matrix from a microarray data set.

As can be seen in figure 5.3.2.3 the technique only picks up the higher level terms associated with DRD4.

[6]This is the same access mechanism used by the other persistent storage components in the system: GOS and GNS, as described earlier in. Arrayexpress2FCAns is designed to be used via the shell function in 26, which is similiar to the g command as used by GOS.

```
% whatis ax
fn ax {
or {ftest -e '#sARREXP/ARREXP'}
   {echo no gene expression data loaded;break}
or {ftest -e /info/ARREXP} {bind ''#sARREXP/'' /info};
q=$*;
rowheader='Affymetrix:CompositeSequence:'
q=q^$rowheader^$q
regextoignore='^disease'
<>/info/ARREXP { echo -n $q >[1=0];
  cat | grep -v $regextoignore | \
  sed 's/''^$rowheader^''//';
  }
}
% whatis axsubset
fn axsubset {
{or {ftest -e ''#sARREXP/ARREXP''}
{echo no Gene Expression data loaded;break};
or {ftest -e /info/ARREXP} {bind ''#sARREXP/'' /info};
q=$*;
rowheader=Affymetrix:CompositeSequence:;
q=c^$rowheader^$q;
<>/info/ARREXP {echo -n $q >[1=0];cat}
}
```

Shell Interaction 26 – *To use the arrayexpress2FCAns service the ax function is used. It takes a gene id to the arrayexpress2FCAns service, which then returns a context matrix for that gene. In addition the axsubset function when given a set of gene probes returns the numerical data associated with those genes.*

```
% cat E-GEOD-2152_614767280_2032433448_2914 | \
 os awk -f $emuroot/$home/data/pull.awk  | \
arrayexpress2FCAns -l 0.25 -d 0.3
% ax HG-U133A:208215_x_at
CompositeSequence GSE5392GSM123228 Norm/VALUE ...
HG-U133A:208215_x_at x x 0 x 0 x x x...
...
```

Shell Interaction 27 – *The signal values are extracted using the awk script. These values are then sent to the arrayexpress2FCAns function which spawns a service to provide binary matrices on demand. The shell function ax then queries this service. The l parameter to arrayexpress2FCAns tells it to list genes that have at least 25 percent crosses with the query gene. The d parameter varies the criteria under which a profile is a match.*

```
% ax HG-U133A:208215_x_at | grep -v 208215_x_at | wc
       7     352    2003
% ax HG-U133A:208215_x_at | grep -v 208215_x_at | \
conceptderive -rn | FCA2fs -ngene
```

Shell Interaction 28 – *Ax run with the gene probe 208215 corresponding to the gene DRD4 finds 6 genes similiar to it. Next DRD4's context matrix is fed into conceptderive and FCA2fs.*

GO ID	Definition
6021	myo-inositol_biosynthesis
16020	membrane
16020	membrane
7268	synaptic_transmission
7212	dopamine_receptor_signaling_pathway
7212	dopamine_receptor_signaling_pathway
7186	G-protein_coupled_receptor_protein_signaling_pathway
7165	signal_transduction
5887	integral_to_plasma_membrane
5886	plasma_membrane
4952	dopamine_receptor_activity
4952	dopamine_receptor_activity
4952	dopamine_receptor_activity
4872	receptor_activity
1584	rhodopsin-like_receptor_activity

Table 17 – *The GO terms of DRD4.*

5.3.3 Results

5.3.3.1 Deregulation of genes under acute myeloid leukemia

Acute Myeloid Leukemia (AML) is a cancer of the myeloid bloods cells, characterised by a rapid increase in the number of mutant white blood cells. Whilst AML is a relatively rare disease it is the most prevalent form of adult leukemia. An incidence of AML can be classified by the chromosomal aberration of the leukemic cell, this in turn offers a strong predictor for survival and relapse rates. In the paper[42] on myeloid leukemia the authors Gutiérrez et al. used gene expression in order to attempt to establish significantly up or down regulated genes that could be used to diagnose the particular class of AML. In their work they discovered 23 genes (reported in the abstract as 21) with 26 associated probes that were sufficient to assign AML to one of these three classes: APL, inv(16) and any other AML subtype without a specific translocation. While the conceptontofs system with the order preserving arrayexpress transducer coud be applied to every gene and the results filtered to identify clusters of genes, this would replicate the functionality of the order preserving method described in [65]. Whilst this is a useful task it would not demonstrate the key contribution of this work: the ability to integrate specific knowledge derived from data with general knowledge aquired from an ontology. Instead the attention and discussion is focused on the labellings of the unfolded concept lattices of a handful of genes.

5.3.3.2 Query gene probes

5.3.3.3 Parameter Estimation

In order to create useful concept lattices and directory structures it is neccessary to restrict the number of matching genes to less than 20, any more and the concept lattice becomes too large to be calculated (or navigated). The way to control the number of matching genes is via the l (support) and the d (delta) parameters that are given as options to arrayexpressns. By decreasing the l value and/or increasing the d value this loosens the criterion under which two genes are matched. There is a tradeoff with a more stringent criteria failing to account for noisy data and too loose a criteria matching arbitrary genes. In table 18 the average number of matching genes for a selection of 30 genes in addition to a query gene are plotted for varying values of l.

		d								
		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
l	0.2	81.40	1006.63	3061.90	5657.83	7873.57	10632.30	11349.83	13008.70	14351.93
	0.3	0.67	37.67	338.67	1217.37	2462.73	4431.07	5339.80	6766.60	7902.53
	0.4	0.00	0.33	13.07	121.77	475.73	1378.23	1981.40	2522.40	3160.87
	0.5	0.00	0.00	0.03	1.23	23.00	174.30	232.47	393.83	591.30
	0.6	0.00	0.00	0.00	0.00	0.47	9.73	10.60	37.07	75.00
	0.7	0.00	0.00	0.00	0.00	0.00	0.13	0.10	0.70	2.43
	0.8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 18 – *The average number of matching genes for a representive set of 30 gene probes after varying l and d. Raw expression data taken from [42].*

5.3.3.4 Analysis

Figures of some of the annotated directories for various genes are depicted in figures 23 to 26. Note that each arrow depicts a different path between two concepts defined by the path that is taken from the root concept to the parent concept in the pair. The experiments and genes are not driectly labelled in the figures either. In the Figures 23 and 24 are depictions of the same gene but from different probes one based on an exemplar sequence the other on a consensus sequence. Interestingly neither probe was coherent with the other and they do not share any coherent genes either. Figures 25 and 26 show more complex directory structure.

In the assignment of GO terms to links the labelling algorithm succeeds in findling labels for a high percentage of the links are labelled (in the examples shown and with other comparable sized lattices between 50% and 70%) when the size of the underlying concept lattice is manageable. Unfortunately with larger lattices the algorothim finds labels for a far smaller percentage. The only way to check the accuracy or appropriateness of the labels would be by surveying the opinions of experts.

5.4 Summary

In this section some the application of the ConceptOntoFs file system to various sources of data was described. Including the Audubon Society Field Guide to Mushrooms, a dataset frequently used for benchmarking . The id3 based music file dataset, which demonstrated some of the mechanisms that are possible with per-process file mechanisms. A simple source of gene experimental information (the L2L data set). Finally a sophisticated technique based on the coherence of genetic expressions that yielded a large source of gene and experimental conditions context tables. For the latter two cases it was possible to utilise the Gene Ontology to provide names for the concepts that were found. Whilst the process of assigning names to the concepts works, the quality of the tags has not been investigated. To do so is beyond the scope of this thesis, though it is unquestionably an important and neccessary task. A way in which that could be accomplished would be by asking a group of molecular biologists to assign ratings to the derived labelled concept lattices. Part of this is due to the decision to reimplent the concept derivation part of the system, it would have been easier and less time consuming to use a preexisting system.

There is an obvious next step that this work can take, by merging the content of *every* microarray experiment stored within the Arrayexpress database to generate a directory structure for every gene. In order to do this a sophisticated transducer that could incorporate the various clustering techniques and the data provided by Arrayexpress would need to incorporate more information about the experimental conditions (for instance if two experimental conditions are replicates and should be averaged).

Whilst the system shows considerable flexibility in the forms of data that can be handled and the functionality that can be plugged in to produce context tables, it is unarguably a prototype system unsuited as yet for widespread usage.

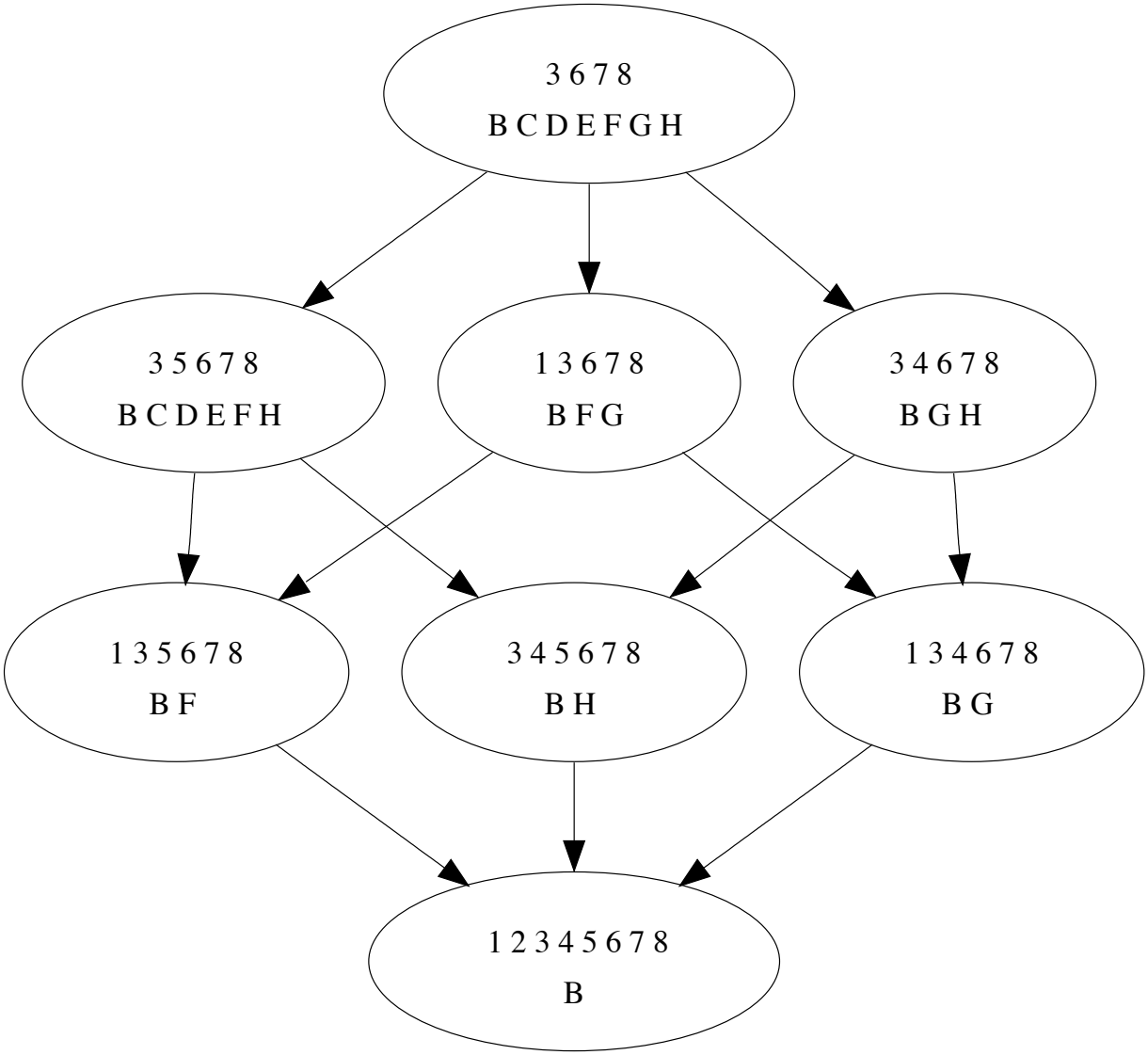


Figure 21 – *The concept lattice of gene B.*

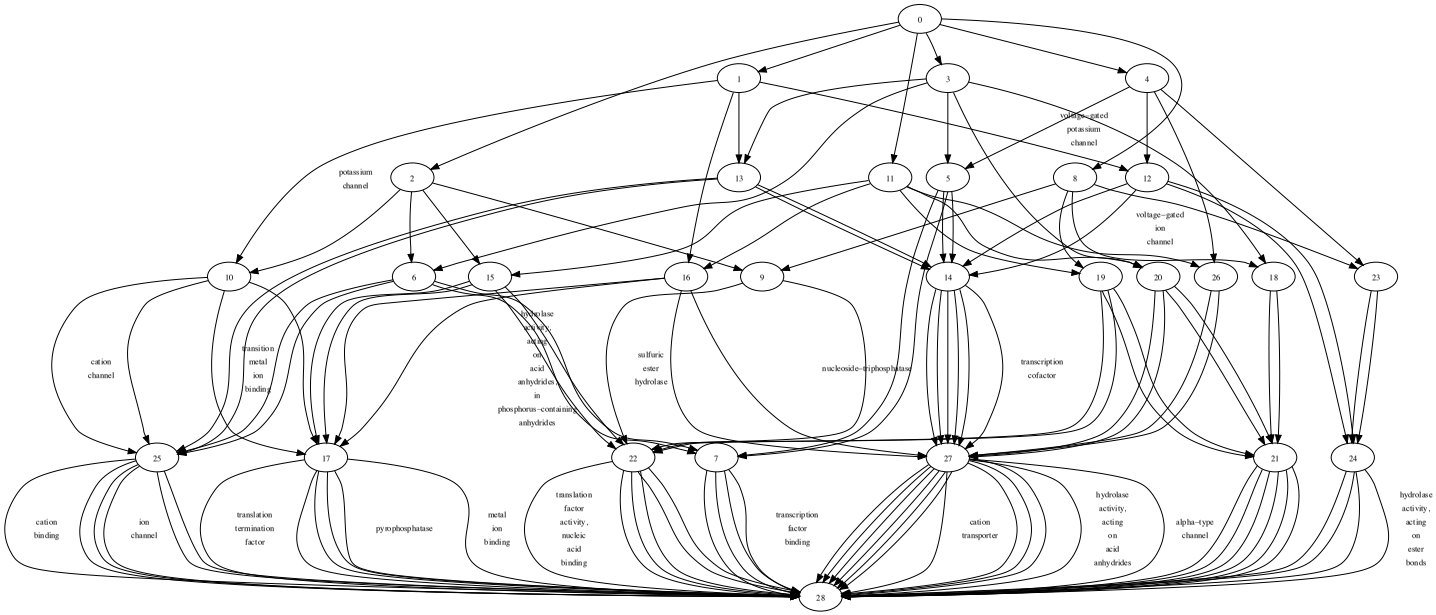


Figure 22 – *Concept lattice of similar genes to the DRD4 gene. In system only finds labels for some paths.*

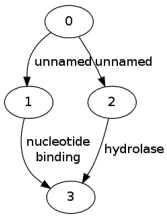


Figure 23 – *Directory structure of the 201495_x_at gene probe for the MYH11 smooth muscle myosin gene, based on the consensus sequence. The other probes matched were 208692_at (ribosomal protein S3) and 213583_x_at (eukaryotic translation elongation factor 1 alpha 1). Parameter values: l=0.5 d-0.4*

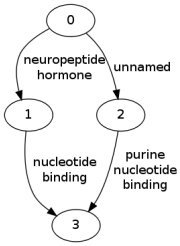


Figure 24 – *Figure for gene probe 201497_x_at for the MYH11 smooth muscle myosin gene(exemplar sequence). The other probes matched were 213033_s_at and 214040_s_at l=0.3 d=0.2*

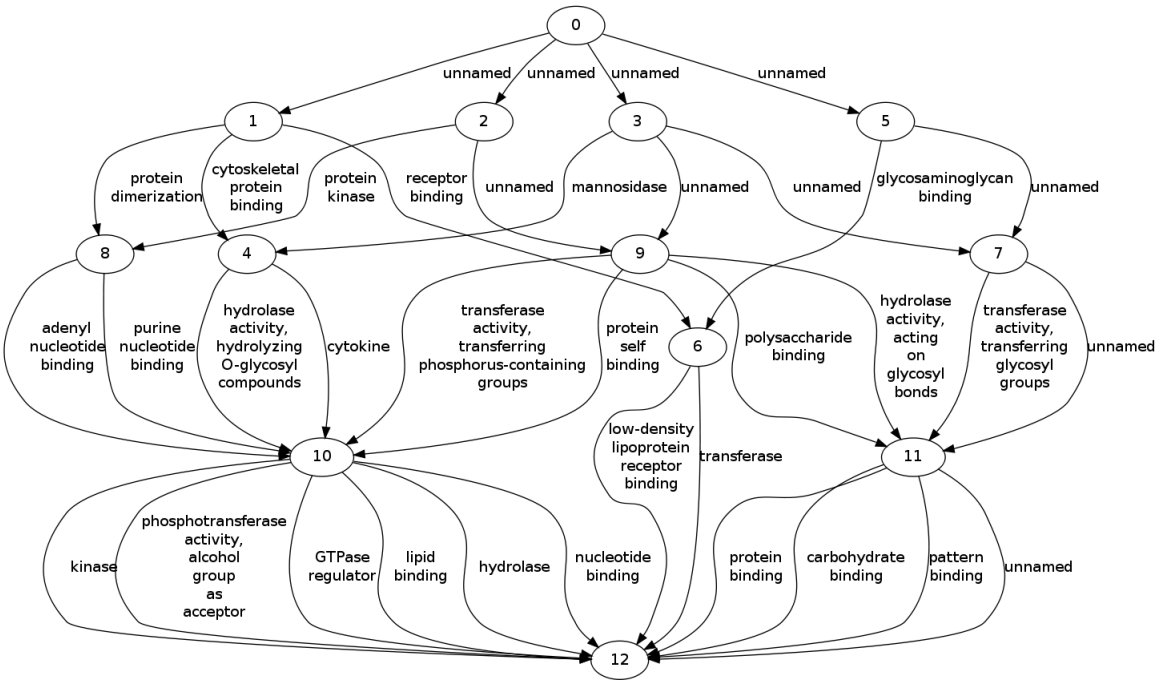


Figure 25 – *Labelled file system for vascular endothelial growth factor 210512_s_at*

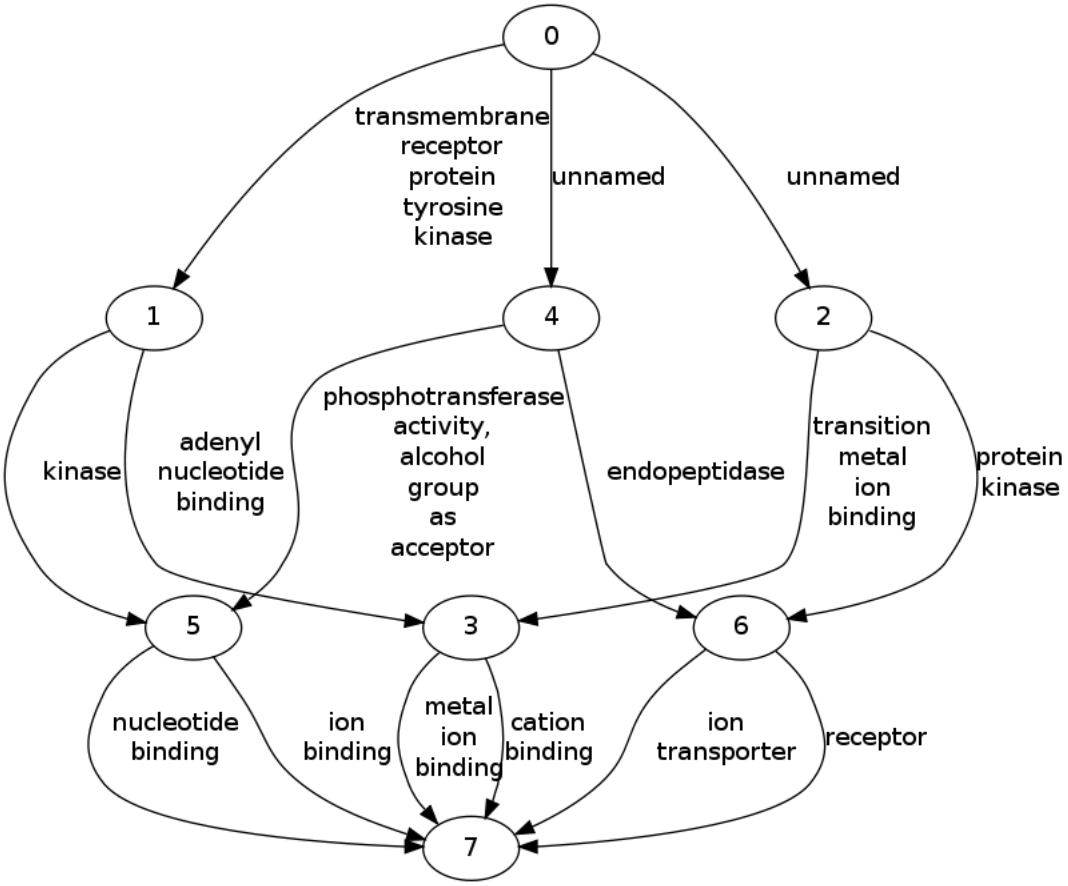


Figure 26 – *Labelled file system for fibroblast growth factor receptor 1 211535_s_at0.30.4*

Chapter 6

Reductionism: structure of biological function.

In this chapter a deeper challenge inherent with transforming the data about complex molecular biological systems into knowledge in the form of models of the behaviour of the system is explored. The principal theory underlying much of biology is evolution. Evolution does an excellent job of explaining why an organism may have come about however it does not explain what organizational functional requirements are necessary for an organism to remain living. In particular the question of assigning functional labels to components (such as genes) has revealed a tension in reductionist approaches to biology. Some of the simple assumptions made earlier in the thesis about what a gene is (such as only considering its molecular function) and conceptual approaches to modelling it can now be more closely examined.

A hypothesis that is considered is that there is no global privileged or unentailed level of causality within living biological systems. The ramifications of this hypothesis are explored, including the motivation for integrative non-reductive systems biology approaches. Further in this chapter a small closed causal system - first developed by the theoretical biologist Robert Rosen's - is analysed. Closed causal systems in particular highlight a difficulty in assigning function to components. With this small system an alternative functional ontology whereby a system is represented by the processes that occur as a result of the functions rather then the objects that are the agents of the processes is explored. It is postulated that there are some deep organisationally invariant effects that are best expressed via some method of abstraction.

There is still considerable controversy over whether Rosen's mixture of philosophy and mathematics adequately demonstrates his argument, this suggest that at present even if Rosen were correct he failed to elucidate a completely watertight proof. However neither can it be said that his detractors have completely proven their objections. [1]

[1] For example the Chu and Ho paper [17] suggested that a cellular automata based universal constructor would be sufficient . However since the rules underlying the cellular automata are not being generated by an entity existing

Rosen's reasoning runs counter to the main argument of reductionism which holds that complex entities can be understood in terms of the sum of the behaviour of their simpler atomic constituent parts. This essentially relational view is in the case of Rosen's (MR) system defined by everything that happens inside a living cell in the context of a functional organisation that makes self-fabrication possible.

The principal thrust of this chapter is to look at the definitions necessary to encode an abstraction of a living system, to examine this abstraction and finally to speculate how similar constructions could be used to develop better function based ontologies for systems biology.

6.1 The challenge

It was briefly mentioned at the very beginning of the thesis that whilst there already exist large volumes of data, however still more data is required. This partly necessitates new kinds of experimental methods and data gathering techniques. The temporaral resolution necessary to infer certain data structures such as gene regulatory networks is an order of magnitude more then is currently available.

Even if the need for higher quality data was matched there still remain major conceptual challenges to overcome. In the paper [21] the failure of genome sequencing so far to provide breakthroughs commensurate with the effort and enthusiasm invested upon it is blamed upon the fallacy of the premise that complex biological organisms can be understood in terms of the individual components that the organism consists of. The incapability of a reductionist doctrine is apparent in many levels in biology, for example whilst it is possible to study a single nerve cell or neuron, and in principle simulate and understand everything it does, in isolation studying one cell may not tell us that a network of nerve cells can give rise to such striking phenomena as the ability to learn.

Is the reductionist goal of understanding biology by consideration of the behaviour of isolated constituent parts possible? Certainly portions of living organisms have been successfully reduced to their constituent components and processes (the heart for instance[82]) however in the context of molecular biology one particular dominant form of reductionism - *genetic determinism* - has had some notable failures. For instance several experiments (for example [48]) have contradicted the previous assumption that a particular genes function could be ascertained by observing phenotype after selective deletions of a gene.

6.1.1 Genetic program

One particular manifestation of the treatment of genes as direct agents is to equate sequences of DNA with program code and the ribosome as computer. While this machine metaphor

within a cellular automata, it is not completely closed to efficient causation. Even now attempts at building and running a Von Neumann universal constructor upon a cellular automata have not been particularly successful.

serves a certain pedagogical purpose it subtly reduces awareness of the capabilities of organisms. For example Alper[6] et al successfully managed to improve the ethanol tolerance of yeast by altering the transciption machinery of gene expression rather then trying to identify and over-express all the genes that would be necessary to elicit high ethanol tolerance. This experiment in addition to several others adds weight to the inadequacy of the reductive - DNA as a program, ribosome as Turing machine - metaphor. DNA is better considered a database, while nonetheless an important component, it does not have a privileged role over any other part of the system. The improvement that was engineered in the experiment by Alper et al is the equivalent of reprogramming the system - without altering the DNA - but there is no easy way of obtaining the set of instructions for the system. The plans for an organism are not easily distinguished from the process of carrying them out.

A principle end result of employing only a reductionist paradigm is the mistaken procedure underlying creating gene ontology. Genes have often been named by the first higher level functional behaviour they are found to play a role in. However as mentioned in section 3.1.5.1 many genes are modular and play a role in multiple processes and behaviours.

There is also an argument that epigenetic approaches [11] where environmental induced developmental changes can also become stably inherited within subsequent generations, provides a strong alternative causal explanation for many inherited attributes. With an epigenetic approach the dynamics of the interaction between development and environmental changes constrain the possible paths of evolution that an organism can take.

Thus possible organism transformations in response to environmental stimuli can be well defined, from this a *taxonomy of biological forms* and a natural system of classification based on the dynamics of processes that generate the forms ([45]) rather then using phylogeny. This method can assist in understanding and representing the functional activity of a system whilst also providing an additional mechanism with which to categorise organisms. The idea of generalising the construction of a taxonomy is further explored in this work [49]. In short phylogenetics is not the only means of categorising organisms or their attributes.

6.1.2 Gene expression and metabolic behaviour

With gene expression data sets it is possible to establish that in a given environmental condition a set of genes is upregulated however the work of linking gene with the resultant behaviour of an organism is still not clear. The expression of a gene is influenced by multiple factors including the environment, other expressed genes and transcription factors. The expression of each gene corresponds eventually to the transient change of the molecule concentration of some entity, this in turn can influence the expression of other genes in addition to causing more direct metabolic changes to an organism.

Knowledge of the gene and the genome is far from sufficient to understand the behaviour and attributes of an organism including even inherited attributes, what are the additional pieces of information necessary? For an isolated gene sequence we do not have reliable

methods for undertaking all of the steps involved in deducing a phenotype from them [20].

For other cases as explained in [21] the process of deriving this knowledge consists of multiple stages and is a order of magnitude more difficult then sequence analysis. The process of going from a list of genes into a list of enzymes has a high rate of failure, perhaps only 50% of the genes have the correct enzyme associated with them. A list of putative gene products, or even a list of putative enzymes, is not a phenotype, and converting it into a phenotype requires construction of a plausible metabolic map. This is complicated since each gene (and its products) may be involved in multiple processes. Since the gene (and their putative products) affect each other with a network of interactions that has a power law structure, the more interesting and relevant a gene the more likely it plays a role in multiple processes.

Finally, the possible phenotypes identified by the metabolic map, can only become a real observable phenotype when all relevant kinetic properties are taken into account, together with information about how all the components are organized into a three-dimensional whole. Temporal effects and the order in which components are made may also play a part. Both the regulation of genes and of the existing metabolic environment alters the eventual behaviour of any sequence of genes - under any given environmental conditions, different genes are regulated to different extents.

6.1.3 Systems biology

If there are issues with reductionist approach to biology - like treating genes as causal agents - then are there alternative or supplemental approaches? One methodology that has been gaining prominence recently is integrative systems biology, an emerging field of research that attempts to provide system level understanding of genetic or metabolic pathways by investigating relationships and interactions of RNA transcripts, proteins and metabolites. What then are the key differences and principles underlying integrative systems biology from reductionist biology?

It is not the reintroduction of vitalism or similar ideas (e.g. 'orgone' energy) where a mysterious non-physical entity is added to matter in order to transform a living system. Instead it is the emergent behaviour that arises as the result of the interactions of a number of mechanisms. This is the ability of nerve cells transplanted from their original location to a new location to take on the structure and function of their new location [107]. These properties are possessed by a system of cells acting rather then any intrinsic function that can be derived from analysing a cell or its components in isolation.

Going further, a key mechanism by which evolution operates - the ability of an organism to function with small mutations - has a viable probability of occurring only if the system has the adaptability that complex multiple interacting relationships foster. With a self replicating entity a small favourable evolutionary mutation to a single component could affect a multitude of components each of which would need to be able to cope with that change. Unless the entity had a means to facilitate this it could result in the failure of the entire entity.

With a living system a single mutation in a gene may increase the rate of expression of a protein which may in turn trigger a pathway that in turn will change the organism. However because of the organizational causality dependence of the other components in a biological system, it will still be able to continue to function. This form of complexity (and resulting adaptability) of an organism is certainly a neccessary condition for a self replicating entity to be capable of evolving .

Thus the focus of investigation is the complex organizational and causal relationships and patterns rather then the constituent physiochemical material components of biology.

6.2 Methodology

Complex is an apt description for the interacting constituent parts of an organism. There are varying approaches to the definition of complex, the definition adopted by Robert Rosen is that a complex system is one that is closed to (efficient) causation. This definition is interesting for its succinctness, its focus on causality and the conclusions that have been drawn from it.

The closure of cause as depicted by Rosen is an emergent property, and from this he derived a pattern or model for a system that he considered to be a basic abstraction of a natural organism. Continuing in the next subsections by considering his tools and his approaches to modelling a complex system.

6.2.1 Causality

The fundamental base for Rosen's analysis of living systems was to consider the essential property of self maintenance and self regulation. In particular to rigorously define these properties in terms of a causal self entailment. In modelling natural systems causation can be considered as a relationship between changes in the state of a system. There are four different forms of causal or (to make) relationships according to Aristotle. Whilst most interpretations depict these relations as causal Professor Marc Cohen[2] defines each one in terms of the verb to make:

- Formal cause: What makes (in the sense of 'what is it to be') an organism?
- Efficient cause : What makes (in the sense of produce) an organism?
- Material cause: What is an organism made out of?
- Final cause: What is an organism made for?

[2]From his lecture notes on philosophy http://faculty.washington.edu/smcohen/320/4causes.htm

Final cause is not perhaps the domain of science and is a question of telogy. Suffice to say Rosen did not want to make any appeal to final cause (several thousand years of religion have tried and failed to make much with that particular relation).

The distinction between material and efficient causes in some cases disappears. For example within a formal lambda calculus system, material and efficient cause are indistinguishable. An algorithm is both a function (efficient cause) and a stream of data and is algorithm as a function. Program and data are treated as the same entity. However real silica systems are not formal systems and in the next section a self-referential security exploit is discussed which demonstrates a difference between efficient and material cause.

The mathematical approach Rosen takes to analyse these different forms of causality is to use category theory rather then the more traditional Russell Whitehead base of sets, functions and relations.

6.2.2 Category theory

The same problems and limitations related to reductive analysis are found in other disciplines besides biology. One simple general example is the Moebius band, its property of being twisted cannot be localised: it is a global property which is independent of either the constituent material or even the local form. Within mathematics - and specifically differential geometry - the method devised to treat such surfaces was algebraic topology, this in turn spurred the development of category theory.

Category theory was deliberately created to provide a tool for dealing with multiple seemingly different mathematical structures, which nevertheless do possess with a greater level of abstraction some commonality. By providing a universal algebra that can capture much of the meaning in different mathematical branches category theory can thus reveal deep relationships between disparate structures. With this abstraction comes an additional benefit, it is possible to derive and express systems that have higher levels of complexity. Category theory is focused on the (structure preserving) mappings between objects rather then the objects themselves. A full and complete introduction to category theory can be found in [57]. Category theory is primarily concerned with links between entities in the next subsection is introduced the way Rosen abstracted a natural system so that it can be analysed by category theory.

6.2.3 MR system

The principle construct that Robert Rosen introduced to abstract properties from living systems is the (M-R) system or diagram. An (M-R) system is best described by considering the *causal diagram* that describes the relationship or mappings between objects in a category. In a causal diagram relationships between two objects take two forms: material cause or efficient cause. In figure 27 solid arrows represent material cause and the source of a solid arrow

is the material cause of the target. Thus according to figure 27 the material cause of B is A (B is made from A). The dotted arrows represent efficient cause, however whilst the of B is f. Rosen

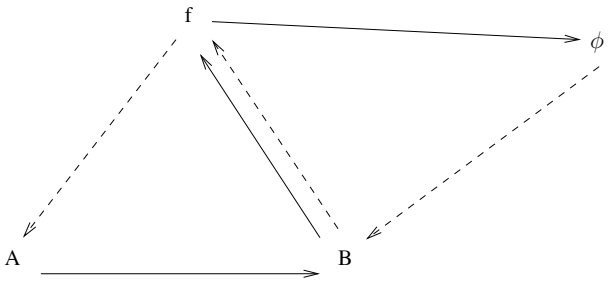


Figure 27 – *An MR system.*

6.2.3.1 Logical impredicative: the self referential compiler

One example of an equivalent situation to the logical impredicative indicated in Rosen's work can be seen by examining a security exploit first expounded by Ken Thompson[111]. First let the objects in the category be some formal data system. Programs are both data and can be evaluated and so can take data and produce other data.

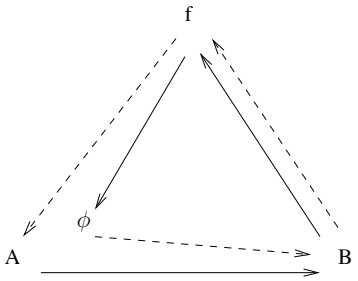


Figure 28 – *An MR diagram with A and ϕ deliberately placed closer together.*

If the function f is a program such that it can take data from A and produce data in B. Every different morphism between the data category objects A and B corresponds to a different program f. In order to have created the system the function f must have been compiled. In order to maintain the system it may need to later be recompiled. Thus the map ϕ must produce program code. ϕ (and indeed f) in such formal systems could be readily identified with A and indeed doing so to Rosen's (MR) diagram yields the diagram in figure 13. It does appear in some domains - for instance the realm of lambda calculus where data and code are equivalent - then there is no distinction between formal, efficient and material cause. However real silica devices can still demonstrate behaviour that shows that this distinction is however still present within them - illustrating that they are instantiations of formal systems and not formal systems themselves.

Let us assume that the function f is a compiler and that the efficient cause of any object can be found by examining the source code. What happens in this scenario is that a compiler is compromised such that it injects a security flaw into login code. Such a security flaw could easily be identified by looking at the source code of the compiler, however in this case a second source variant of the compiler is compiled which given the clean source code of the compiler injects the compromised compiler code into the compiler. Thus a part of the system comes from outside the system. From within the system it is impossible to discover (without outside knowledge that the system is compromised.) A possible solution would be to decompile the given object code in order to determine and be able to inspect the source code. However if the decompiler was created by the compromised compiler then the decompiler too can be compromised.

The external introduction of the unknown exploit is exactly an efficient cause (by Rosen's definition) that the system is not closed to. If the system were closed to efficient cause then this exploit would not exist. It is this form of impredicativity that Rosen questioned, though he postulated that a such a special (one hopes) situation with respect to computer systems is actually an essential characteristic of a living system. This then renders such systems unsimulable within formal systems. Implying there may be some part of a living system that is inherently "hidden" within its structure and form, which cannot be discovered even if one appears to know everything about the mechanisms and physical rules that seem to govern it.

6.3 Examples of self effecting entities

Rosen never provided any actual worked through examples of his (M,R) systems. In his place there have been a variety of examples given by various authors. In this section a brief review and interpretation is made of these techniques.

6.3.1 Ribosome

It has been pointed out many times in the literature that ribosomes are perhaps the best known examples of Von Neumann constructors [104]. They fit the description perfectly: on its own a ribosome can do nothing, but with the information embedded in a messenger RNA molecule that has been transcribed from DNA it can (with the aid of multiple other components such as auxiliary enzymes, co-factors and an energy source, GTP) arrange amino acids together in the specified sequence to eventually produce . However, as Hofmeyer points out the ribosome constructs *only the components.* Even then the components need to fold and then self assemble. As Hofmeyer states "The existence of the non-covalent, supramolecular processes of folding and assembly therefore forces us to search for their efficient causes inside the system." In other words Hofmeyer does not consider the ribosome alone to constitute the efficient cause of the system.

6.3.2 Functional organization of a biological cell

In this work [117] Wolkenhauer adapts Rosen's (M,R) system and used it to construct a formal model of the functional organization of a biological cell. The semantics of his cell model are different from Rosen and the crucial unit of Wolkenhauer's model is a pathway where:

"A pathway is a subsystem whose behaviour is determined by a finite set of variables (proteins, genes). As a subsystem a pathway always has a context or environment with respect to which its behaviour is defined. The boundary between a pathway and its environment is defined by a set of input and output variables. We hereafter consider a pathway to be a network of biochemical reactions, which can therefore be understood in terms of the processes it can realize."

This is done by finding canonical representations of biological (typically past determined strongly non-anticipatory) systems with a constant maximally reduced state space [118], that is he represents a cell process as the temporal evolution of a state and equivalently the state of a system provides a compact representation of the past behaviour of the system.

Wolkenhauer argues that the ability of a cell to self-organize its biochemical processes is equivalent to the (M,R) process that Rosen described. By applying a similar argument to Rosen's he argues that a requisite of any attempt to formally model a self organizing system such as a cell requires the model to be based on a closed Cartesian category.

[3]

[3]These are a particular subclass of categories. A member of which is referred to as a CCC (Closed Cartesian Category). A CCC is a category that fulfils three axioms. A terminal object, the existence of an evaluative or exponential map and a Cartesian product for all objects in the category.

The crucial property that this leads to within a CCC is that the evaluative map is a bijection from $H(X \times Y, Z)$ and $H(X, Z^Y)$, and hence the inverse is well defined. This property allows an efficiently cause closed system with a CCC to have no infinite regress.

Whilst the Set category and the Cat category are CCC. The category of smooth manifolds and smooth maps and the category of all vector spaces over some fixed field are both not CCC.

In particular every object including the higher coordination of cell functions must lie within a co-domain (thus have efficient cause) from within the system. However a model based on such a system would need to be a closed cartesian category, however conventional simulation strategies based on differential equations do not fulfil this requirement.

6.3.3 Autopoiesis

Autopoeisis[113] also concerns the dynamics of living systems, with the main focus of enquiry being the characteristic organization of living systems and the necessity to enclose them with membranes.

There are many points in common between autopoiesis and Rosen's approach, certainly both take an abstract stance and use constructs based on dynamics and relationship rather then focusing on the constituent components. In common with the (M,R) systems autopoiesis does not try and deal with or use reproduction or evolution to explain its approach. Both approaches emphasize the importance of organizational closure, but in slightly different ways in brief, an autopoietic system is a network of processes in which any given process produces components that are eventually transformed by the rest of the network into the components transformed by that process.

What then are the differences? Autopoetic systems only require a material cause closure. Efficient causal closure in the sense of Rosen is a more general form of closure, thus as Letelier et al[59] posit autopoietic systems may be a subset of (M,R) systems. Additionally within autopoetic systems greater emphasis is placed on the membranes (the interfaces) between components. Likewise Rosen has greater concern with the *logical* organization necessary with a self-interacting system.

6.4 A causal taxonomy

MR-systems by virtue of their organization reveal the key problems that underlie assigning ontology to portions of a biological system. With a causally closed system it is impossible to firmly delineate where the influence of one component begins and ends. This can be clearly seen in Rosen's (M,R) system where no component A, Bf or ϕ can be considered to be the sole cause of any process. Certainly by looking at an isolated fragment of a system, it appears as if every process has a well defined cause, however with the insight of being able to see all the processes it is quickly apparent that there are only loops. The same is true in biology where no one level is the sole cause of the behaviour of a biological system[83].

Consider again if a mutation in a gene led to an increased production of a protein

There is a huge difference between the processes which are the realizations of functions and the objects which are their bearers, thus gene ontology as originally conceived by geneticists may never fully be realized [81].

6.5 Concluding remarks

It is easy to see why some critics have been critical of the central result Robert Rosen - the biologist - suggests about living systems. He hypothesized that living systems possess a property that mechanisms or algorithmic processes do not. He was motivated by the proof that there exist uncomputable numbers, that is numbers that are well defined but cannot be expressed or calculated by a Turing Machine or a deterministic algorithm. He postulated that there are mechanisms that are uncomputable - that is inexpressible or unsimulable by a Turing Machine. Then starting from an abstraction of a living system, he described how such an abstraction results in an entity that is uncomputable. He thus conjectured that a living organism, with a similar logical impredicative about it is, thus unsimulable. His treatment of universal entities is far closer in spirit to philosophy then biology or even mathematics.

Even his fiercest supporters would grant that his explanations and proofs require some additional commentary and exposition. Whilst it is unnecessary for every proof to resort to the most technical and formal basics the level of hand waving and bluster in Rosen's exposition is uncomfortable. His style of writing : adopt a somewhat unclear way about big ideas, has produced a body of work that seems tantalisingly attractive. It is hard not to question some of his precepts and he produces no easy answers. On the one hand it seems tempting to dismiss his ideas as negative rhetoric against the ability to replicate living systems. However he never says that artificial living systems are impossible to create. Or even that a living system is not possible to model in silico. He instead states that a living system is not *limited* to what an algorithm can compute.

The hypothesis put forward by Robert Rosen (and later refined and reworked by Wolkenhaur) is that in-silico physical simulation and replicas of biological entities miss a crucial part of mathematical modelling, which is not to mimic biology - no matter how far the physical detail is improved or the scale. Instead Rosen and Wolkenhaur look upon mathematical modelling as a process by which understanding of a system comes about. In this sense mathematical modelling is as much about the *art* of transforming from a natural system to a formal system as the final formal description of a system. What is sought is explanation and understanding rather then description. Rosen's work still attracts controversy partly because his particular method of explanation involves some ambiguity and requires some dedication on the part of the reader. This is unfortunate as it seems a principal driver for debate about his work stems from confusion over his choice of terms or explanations[117, 18, 66]. In addition his central proof whilst possessing logical coherence lacks experimental verification.

However Rosen's work encapsulates many of the challenges that are becoming apparent in systems biology, unfortunately he does not provide ready solutions. Instead his work provides a framework to explore systems biology with a focus on the mappings (potentially the processes) between entities.

The counter point to the other work in this thesis is some of the issues that have been raised in this chapter.

In my own personal opinion the ϕ or reproduce or inverse evaluative map is the key to practical applications of Rosen's work. This map is a material medium dependent entity (by entity it is meant that it embodies the processes, context and physical properties) that permit a living system to persist. An insightful parable by Marcello Barbieri[10] likens a cell to a village or a city, then postulates how these cities might be analysed:

> "The real thing[of study] is the books. It is they that contain the instructions for making objects, and the objects, inhabitants included, were built with the sole purpose of making more books. Buildings were built from drawings of the buildings so that more drawings could be made in their rooms. Telephones were made in order to multiply telephone directories, inhabitants to multiply anatomy books, and so on."

Returning to the original problem, that of finding conceptual bases for working with systemic data there is significant scope for integrating his ideas into data structures like the Gene Ontology and the tools for observing organisms. As has been noted in the literature a category theory approach to [52] ontology has been explored, this is perhaps though only the beginning of what can be done. Category theory is eminently suited to the task of arranging by organization and should be pursued especially in conjunction.

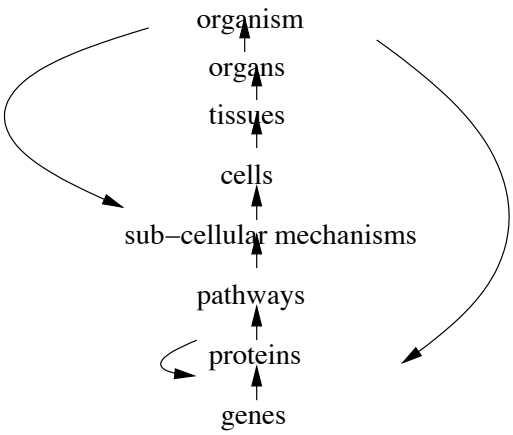


Figure 29 – *A figure adapted from [83] which shows how downward causation applies in biology. The orthodox reductionist approach has been to only consider the causal relationships going in one direction, so that genes cause proteins that cause pathways etc. However there are also mechanisms whereby the conceptually higher up entities influence the lower ones.*

Chapter 7

Discussion

The project addresses the structure and representation of biological knowledge, in the most general form of an object attribute table. Key approach is finding concepts (leveraging off previous work in FCA), the key device is representing them as a file system inspired by previous work.

The approach of this project was to take these ideas and explore the implementation of a knowledge management system using recent developments in distributed computation. These developments use the file system as the key interface to a variety of resources, so that there is a natural mapping to Concept-based file systems. The key technical advance is a fresh implementation using a modern, lightweight, platform independent, distributed operating system, with a component-based multi-threaded dynamic modular architecture. The key scientific advance was to address the issue of how to name concepts, which contain objects with similarities derived over (perhaps many) different attributes. This issue is key to the compact and natural representation of the concepts as directories within a file system - and was thought crucial to its navigability and for deriving insight from the knowledge structure. The approach taken was to name concepts by reference to external ontological structures that the objects have been annotated with. This approach, required some careful considerations of which terms from the ontology were suitable for naming, but offers the promise of compact, domain-specific terms which summarise the derived concepts.

In the specific use case of finding structure in gene expression data sets, the project leverages off the significant effort that has gone into annotating genomes against standard ontologies (the GO). The results from illustrative test cases on gene expression data suggest that the system is capable of deriving concept and placing them in a navigable file system structure. The ontology-based concept naming of directories, proves to add significant value to the meaning and navigability of the conceptual structure.

We note that the architecture is generic. Any data capable of being expressed as an object attribute table can be processed by the system. The current restriction, though easily removed,

is that of binary attribute values. We demonstrate the generic nature of the system via an example file system for navigating music files, built from embedded meta data tags.

The performance of the system is adequate for illustrative purposes, this not being a primary objective of the project.

The architecture of the system is worthy of remark. The system embodies the component-based toolkit approach that first was demonstrated by the developers of the UNIX system. The philosophy here is to build small tools, that do one thing well and that provide simple interfaces to other such tools. The key glue binding the toolkit together is the pipe, and key interoperability through the discipline of representing data as simple text-based data streams. This approach, in which complex tasks can be performed by simple construction of a pipeline of tools is sidelined in the most modern software systems: where monolithic applications provide built-in functionality, rivalling that of some operating systems. The development of Inferno, from the same group who developed UNIX, puts the component-based philosophy onto a new plane. In Inferno (and Plan 9), files can now be arbitrary system resources, and can be mounted on arbitrary points on the network. Inter-process communication (and thus distributed applications) can be achieved by simple reads and writes to files in a per-process namespace. Other advances include more fully-featured shells, which offer more flexible pipeline construction and also the potential for non-linear or branched work-flows. When dealing with complicated problems there is no way of escaping from complicated software. The choices then arise in how to manage this complexity.

This system was built in this new software environment. Although few applications of the system are described in this thesis, it was intended to demonstrate how (large) knowledge resources may be represented as conceptual structures, and navigated as conventional file system. The ready distribution of such file systems suggests a straightforward realisation of the physiology of the semantic web.

There is a wealth of further work that can be built upon this. Performance and scalability have considerable scope for improvement, currently the benchmarks for the principal part of the tool-set show that it is several orders of magnitude away from other similar concept analysis implementations[62]. There are many extensions that are possible: further data sets could be incorporated, further transducers created and likewise ever more diverse scaling techniques could be implemented. All of these could dramatically increase the range of possible applications. The system does not offer a user friendly or GUI interface, and another extension would be to implement a GUI interface. This could be implemented upon the tools so as to increase usability whilst not compromising the flexibility of the system.

In this work I have shown the implementation of a key bioinformatics technique using a novel computational paradigm and architecture. The key missing element is usability. Other implementations use more main stream implementations and therefore benefit from user familiarity, however this work is somewhat different perhaps more elegant in some places, however more obtuse in others. This is partly a reflection of the exploratory prototypical

nature of this solution to data management. One of the consequences of the philosophy that govern its design is that there have been compromises with regard to its usability; looking at it as a tool for biologists to use, it unquestionably requires a significant commitment to use and become familiar with. In part this is a reflection of the corresponding complexity and conceptual baroqueness of modern molecular biology - biology is a complex subject and needs powerful tools.

Appendix A

Inferno idioms

I want to explain exactly how the interaction of components happens in Inferno (many of the following is also relevant to Plan 9 and to a lesser extent Unix).

Whilst an Inferno program can specify that a particular file be associated to a file handle, there are a number of ways that the file (or in turn resource) that is actually being used be altered. This in turn permits a number of idioms about how file server based resources are used.

A.1 File Descriptors

The Inferno Shell always provides all Inferno programs with at least three file descriptors standard in standard out and standard error. They are referred to by number 0=standard in, 1=standard out and 2=standard error. Other files that are opened and accessed get file descriptor numbers that are higher. For a program run from a console standard input, output and error are all connected to the console. The file descriptors open for a particular process can be examined by reading the file fd in the /prog for that process. Each line describes an open file. The fields are: the file descriptor index, the open mode (r, w, rw); the type and number of the device; the path, version and type of the file's qid; the file's atomic I/O unit; the file I/O offset in bytes; and the name with which it was opened.

For example

```
ps | grep WmSh
  70        69      abhey    0:00.0         alt     32K WmSh
% cat /prog/70/fd /usr/abhey
0 r   c    0 (0000000000000001 0 00)      0         0 #c/cons
1 w   s    0 (000000000000000b 0 00)      0         0 /chan/wmstdout
2 w   s    0 (000000000000000c 0 00)      0         0 /chan/wmstderr
3 w   s    0 (000000000000000d 0 00)      0         0 /chan/plumb.input
```

```
4 rw s    0 (0000000000000007 0 00)     0      277 /chan/wmctl
```

this shows that the shell has five files open. Standard input, output and error are routed to the standard input, output and error of the shell window.

A.2 Inferno Shell

Whilst a good thorough introduction to the Inferno shell can be found here, the following is sufficient to make clear the examples in 4.

A.2.1 The Algebra of I/O redirections

Consistent access to file servers is facilitated by the semantics of using file descriptors in the Inferno Shell. Every program started by the Inferno Shell has three file descriptors available to it: standard input, standard output and standard error. Operations (familiar from UNIXshells) include shell redirect to a file from standard output. e.g

```
ps > alistofprocesses
```

or

```
gunzip < zipfile.zip
```

A useful aide de memoire is to consider the arrow as showing the direction of data flowing. In this case whilst the grammar appears to be program | redirection operator | file. However the shell redirection can equivalently appear before the command e.g.

```
> alistofprocesses ps
```

is the same as

```
ps >alistofprocesses
```

This is because the shell parses each block of input, upon encountering shell redirection operator it reassigns the file descriptors then once the entire block is parsed it actually executes the programs within.

What a shell redirection '>' actually does is substitute for standard output (file descriptor 1) another file. Likewise shell redirector '<' changes standard input (file descriptor 0) to another file. A useful feature for long running processes would be to be able to send diagnostic output to a log file whilst retaining normal output capabilities. Shell redirections can be actioned on file descriptors other then standard input and output by the following syntax

```
limbo -w blah.b>[2] junk
```

would redirect file descriptor 2 to the file junk, which would send all error output to the file junk. Note that the -w option in the limbo compiler makes the compiler print warning messages to standard error.

File descriptors are handles that point to files and they can be reassigned by using the dup system call. They can also be reassigned using the square bracket of a shell redirection. For example

```
limbo -w blah.b >[2=1]
```

, would send output meant for file descriptor 2 to file descriptor 1.

- limbo -w blah.b >[2] junk sends errors to a file named junk.
- limbo -w blah.b >[2=1] sends errors to standard output.

It is useful to mentally transform the '=1' part as replacing the file name with a direct file descriptor. Multiple shell redirections may appear in a command block. While it may appear at first that they "are linked together", the semantics is different. The state of the file descriptors is changed by each redirection, the shell processes each redirection from left to right. This means that

```
limbo -w blah.b >/dev/null >[2=1]
```

will send standard input to /dev/null and also send standard error to /dev/null whereas

```
limbo -w blah.b >[2=1] >/dev/null
```

will send standard error to standard output (hence the terminal) meanwhile sending standard output to /dev/null. Only after all shell redircetions have been carried out will the shell execute any commands.

There is one further Shell redirection operator: '<>', It connects standard input of the command it is being used with the file it is associated with (in a similiar manner to '<'), however it is possible to both read from and write to the file. With a regular file system that treats files as bags of bytes this operator appears to be somewhat useless, however in Inferno it permits the following operation

```
<> /net/dns {echo -n 'google.com ip' >[1=0];cat}
google.com ip   64.233.187.99google.com ip      72.14.207.99google.com ip
```

What this does is connect the standard input of the command block to the file /net/dns. Then within the command block the "echo" command opens the file /net/dns and writes the string "google.com ip" and then the "cat" command reads the file using the same handle and prints it out.

A.2.2 Compound commands

A.2.2.1 Piping

Consider the problem of counting how many processes are currently running, one way to do this would be to do

```
ps > alistofprocesses
```

, then do

```
cat alistofprocesses > wc
```

However the file alistofprocesses is temporary and may only be needed for this one operation, it seems unneccesary to name it, create it and delete it. The Inferno Shell provides a mechanism that allows command input and output to be chained together making the creation of temporary files like this unneccesary. The syntax is again familiar from Unix shells:

```
ps | wc
```

It is possible to specify which file descriptors are connected together. For example

```
%limbo -w blah.b
blah.b:17: warning: argument ctxt not referenced blah.b:17: warning: argument
%limbo -w blah.b | wc
blah.b:17: warning: argument ctxt not referenced blah.b:17: warning: argument
   0       0       0
```

Here wc is counting the standard input given to it which in this case is the standard output of the command *limbo -w blah.b,* however this commands prints these commands onto standard error. In order for wc to see these lines it is neccessary to specify standard error to the pipe:

```
limbo blah.b |[2] wc
   2      12      98
```

. Just like in shell redirection standard error is sent to the command on the right hand side of the pipe. It is similiar to doing limbo blah.b >[2] tmp and wc < tmp.

This mechanism is used in [13].

Piping commands which are are all pure functions (i.e. they have no side effects), together in this manner is an example of monadic input output, that is the state of each function is managed solely by its parameters.

A.2.2.2 Command substitution

Besides piping, the Inferno shell provides another mechanism to embed a command within another command. A command within a pair of curly brackets is prefixed with a backquote operator, now the command within the brackets is executed first and the standard output of its execution is read by the shell. For example in order to find the total size of all limbo source files in /appl/cmd that contain the word 'free'

```
cat `{grep -l free /appl/cmd/*.b} | wc
```

The command `grep -l free /appl/cmd/*.b` returns the names of all files that contain free, this list is then passed as an argument to `cat`. Which in turn opens and reads each file in full sending it via the pipe to `wc`.

A.3 Threading and concurrency in Limbo

A consistent and recurring challenge with developing file servers (and indeed many other applications) is handling events in an indeterminant environment. Writing and debugging deterministic programs that handle indeterminancy in input is a non trivial task. Limbo adopts a novel paradigm based on Communicating Sequential Processes (CSP)[46]. Here two ingredients are neccessary, threads and channels. *Threads* are light-weight processes sharing a single address space. Multiple threads can be spawned concurrently. Making a subroutine (it is not allowed to return a value) a separately run thread is trivial in Limbo, simply call it with the keyword spawn, e.g.

```
spawn newproc();
```

Threads allow the proper structuring of non-deterministic applications, however they bring two problems: race conditions and deadlocks. Deadlocks occur when two processes (equivalently threads) are both waiting for each other to complete. They are locked in a deadly logical embrace as neither can finish unless the other completes. Race conditons arise when two threads both try to access or modify a common global data structure, the solution is to put some form of locking or control around all access to certain data structures. Most shared data then needs to be locked before accessing and unlocked after. However locking can be non trivial to get right in all cases. Programmers can mistakenly set or clear locks, not realize a lock is neccessary or set the wrong ones for the data of interest. These mistakes and others, can lead to deadlocks. With CSP [46] Tony Hoare offers a method that allows both of these problems to be detected by logical analysis of a program (without needing to run it directly) provided an additional data structure (of channels) is available to developers.

Channels, provide connections between threads. The channels in Limbo are typed and can be used to either transfer information or synchronise two threads.

The mechanism offered by channels consists of two kinds of operation, data can be sent down a channel or received. When a channel statement appears in a thread progress is halted until another thread accesses the channel. There is an operator for channels '$<-$' , where the source is on the left and the destination is on the right. If c is a channel and b is an integer variable, then to send something down a channel, place the channel name to the left of the operator and an assignment to the right: $c <- = 2$. To receive from the channel place the channel name to the right of the operator and the assignment to the left: $b :=< -c$.

A useful mnemonic to understand the behaviour of the channel operator is to imagine that the arrow signifies the direction of data movement in and out of the channel.

Program 1 This segment of code illustrates spawning a new thread to run the subroutine anotherproc and using a channel to communicate between the two running processes.

```
#This is the main process
init(nil: ref Draw->Context, nil: list of string){
#declares x as a channel of integers
  x:=chan of int;
  a:=int 0;
  #Starts another thread
  spawn anotherproc(x);
  #send 2 to the chan x
  x<-=2;
}
#define anotherproc
anotherproc(y: chan of int) {
#read a value sent down x and assigns it to b
  b:=<-y;
}
```

It is also possible for one thread to wait for any of many alternate channels to become available.

Generally concurrent activity in a problem should be correspondingly represented by concurrency in a program. That is by having threads running for each concurrent component. Since threads in Inferno are extremely lightweight it is possible to have many threads without requiring or using large amounts of resources like memory.

The alternative to using threads and channels

- busy waiting techniques such as semaphores.
- monitors.

A major disadvantage is that it is not as easy to verify safe concurrent behaviour with either of these approaches. With channels a model can be built of the concurrent behaviour of a program and the model verified, tools exist to build and verify models including Spin[47] and FDR.

Typically an Inferno application has one main controller thread, and multiple threads: one for each of the various inputs. Each of the input threads then communicates via a channel to the main controller thread.

A.4 Mounting and binding

Where does the root of the namespace come from? Within Unix the root comes from the filesystem that was booted from. The namespace of each process is assembled from the namespaces of the device drivers in the kernel. It is possible in Inferno and Plan 9 to alter this view without directly effecting the resources acted upon. This is done via the bind commands (and system calls). Since the bind command only binds parts of the namespace with Each device driver provides its own namespace, and has a place in associated with it.

A namespace only requires having a process that is capable of reading and responding to styx messages and the mount command allows this portion of namespace served by a process to be mounted:

```
mount {command} /n/userfileserver
```

Furthermore a portion of the namespace can be exported from a networked resource (it merely needs to use the Styx protocol and need not be an instance of Inferno) and mounted within Inferno. To export a file-system from Inferno the styxlisten command is used:

```
styxlisten tcp!*!1257 export /n/filespace
```

Appendix B

Computer Environments

There are a variety of styles of common user interface that are used for interacting with a computer and data sets stored on it.

B.1 Interaction

B.1.1 Command line based

Commands are typed in and a dialogue is carried out with the computer. There is a syntax or grammar of the usage of the commands together with an underlying meaning behind the commands. A language therefore expresses actions.

B.1.1.1 Interpreter based languages

A program interprets the commands that are typed in. Examples include the /sc unix shell, R and Biolisp[112].

B.1.1.2 Structured Query languages

An important class of interpreted languages are the query languages. Query languages are based on a calculus that is simple, there are however constraints upon how expressive they are and hence the queries they can answer (they cannot for example easily express transitive closure 3.3.3). They are somewhat limited by how many data manipulation operations they can actually carry out.

Structured Query Language (SQL) is built upon these ideas to access and manipulate data. Commands in SQL are parsed and executed by the SQL engine. This form of data access interface, while useful for developers and advanced users, is not normally employed directly by end users. Instead some form of user interface is layered over SQL calls, often

by using a web based interface. This has traditionally been an activity replete with templated (*boilerplate*) code [114].

B.1.1.3 Textual Interface: Acme

The evolution of the command line interface in some computing environments like Acme Oberon or Smalltalk is to allow text anywhere on the screen to be used for issuing commands. There is no wasted screen real estate. Typing a command anywhere within Acme and then selecting it with the mouse 2 button results in that command being executed. All commands to Acme are issued in this manner, Acme has a relatively small command set augmented since any command it does not recognise is passed to the shell for execution. In this way the textual interface retains much of the functionality of the command line. While there are a number of commands built in to Acme, here I only highlight the Edit command. The Acme Edit command treats the argument given to it as a text editing command in the style of Plan 9's Sam editor.

In addtion a mouse button chording system described within the man pages:-

> "Several operations are bound to multiple-button actions. After selecting text, with button 1 still down, pressing button 2 executes Cut and button 3 executes Paste. After clicking one button, the other undoes the first; thus (while holding down button 1) 2 followed by 3 is a Snarf that leaves the file undirtied; 3 followed by 2 is a no-op. These actions also apply to text selected by double-clicking because the double-click expansion is made when the second click starts, not when it ends.
>
> Thus to copy a word a number of times, double click on the word with button 1 to highlight it leaving button 1 down, press and release button 2 to cut it and save it in the snarf buffer, press and release button 3 to paste it back and then release button 1. Now move the cursor to any selected place in the text, press button 1 down, then button 3 and the word is copied in."

This use of mouse buttons in place of the more familiar copy (modifier+c) or paste (modifier+v) is hard to describe however completely intuitive in use. It physically feels as if a selection of text is being picked up(pressing button 2) and then dropped (upon pressing button 3). It is possible to override the default functionality of the Acme editor by providing programs that handle acme events.

One of the major issues with Acme is that the learning curve (effort in the x axis and reward in the y axis) is shallow. This means that some effort and perseverence needs to be expended before its utility becomes clear.

B.1.2 Graphical User Interfaces

Graphical user interfaces enable human computer interaction by, employing a set of visual objects that can be manipulated with a mouse or other pointing device, to represent the actions possible by the user. There are many frameworks that allow building consistent interfaces for applications.

B.1.3 Discussion

There are then two main approaches with advantages and disadvantages, have a graphical user interface that is easy to use but not scalable over large data sets. Or provide a command line which has a steep learning curve but permits more flexibility and scriptable behaviour. Many standard bioinformatics tools including BLAST [7] are accessible in two ways: as a website (thus providing a simple GUI) and as a command to be used on the command line.

Appendix C

File Orientated Software Engineering

C.1 Files as Objects

Files in the Inferno and Plan 9 world are not simply bags of bytes. They are objects. There are five fundamental operations that can be carried out on files:- create, open, close, read and write. Writing instructions to a ctl file can cause actions to be performed. Upon file systems there is the further operation of attach and navigation. This is analogous to the fundamental operations available on objects. Instead of object hierarchies there exist file system hierarchies. More fine grained access to private and public data and functions of an object can be done with file system permissioning.

There are two types of hypothetical Object Orientated programmer - the ones that create the object libraries and the ones that use the object hierarchy components. In this alternate paradigm there is a distinction between the file system creator (typically using a programming language like C or limbo) and the shell script user (using a completely separate programming language). As a programming paradigm this approach has never achieved popular success partly because of the difficulty (and impossibility in most systems) in developing user space file systems.

Message passing between concurrenty running file system objects is possible via a uniform and consistent interface.

C.2 Design Patterns

Design patterns was originally a term coined in the field of building architecture to describe an abstract set of recurring solutions to the problems that arise when designing a building.

The term has also been adopted to describe abstract solutions to the problems that arise in software architecture particularly in Object Oriented frameworks. Gamma and Helm et al describe a minimal set of design patterns.

With ease it can be seen that the persistent file systems that make up the Plan 9 and Inferno architecture adhere and are exemplars of many of the design patterns from Gamma et al.

Here I describe the design patterns:

- A singleton: trivial property of the namespace, all singleton file servers check there is not already a running instance in the current namespace.
- Factory Method: Clone file system. New directories are only created via the clone mechanism.
- Adaptor: srvssh or execnet
- Facade: wrap a file system around a resources
- Interpreter: Inferno Shell
- Composite: Hierarchical File Tree

Appendix D

Data

Concepts	Objects	Attributes	Terms
0	uvc_ xpcs_ 8hr_ dn uvc_ xpcs_ 4hr_ dn uvc_ ttd-xpcs_ common_ dn alzheimers_ incipient_ up alzheimers_ disease_ up	MALT1	nil
1	uvc_ xpcs_ 8hr_ dn uvc_ xpcs_ 4hr_ dn uvc_ ttd-xpcs_ common_ dn alzheimers_ disease_ up	MALT1 C19ORF7 NFIB CDC2L5 IGF1R E2F5 KLF7 WRN C4ORF8 CENTB2 TIPARP PPFIA1 MARK3	6375 244 387 16540 6739 398 377 375 42623
2	uvc_ xpcs_ 8hr_ dn uvc_ xpcs_ 4hr_ dn uvc_ ttd-xpcs_ common_ dn	MALT1 C19ORF7 NFIB CDC2L5 IGF1R E2F5 KLF7 WRN C4ORF8 CENTB2 TIPARP PPFIA1 MARK3 APPBP2 TRIM33 ZFHX1B PHLPPL UBR2 KLF6 LARP5 RASA1 RASSF3 BDNF DOC1 DOCK4 RUNX1 SLC25A12 HEG1 TRIM2 CDK8 VPS13B TIAM1 EDD1 NAV3 ACSL3 ABCC1 MTAP GLRB IRS1 HOXB2 ASXL1 HRB TSC22D2 MICAL3	6375 244 387 15822 5237 45767 6020 16933 16540 30041 16408 6739 19319 46364 8064 43066 30832 15807 6839 8154 82 398 377 375 6769 51258 8380 4714 7249 7169 5231 51028 19362 19199 6397 16485 6733 42113 6916 30029 30098 16071 50658 50657 6396 6405 30036 19318 30005 51168 6875 45934 15630 15698 16051 6865 6820 19941 4713 46942 45449 51169 6351 44257 51603 7067 6366 16310 46649 45321 30163 6913 6464 6357 17111 42623 6468 16887 6508 6355
3	uvc_ xpcs_ 8hr_ dn uvc_ xpcs_ 4hr_ dn alzheimers_ incipient_ up alzheimers_ disease_ up	MALT1 NCOA3 RBPSUH FYN SRPK2 QKI C13ORF24 ZNF198 ZHX3 RNGTT MYO10 CAP350	4468 398 377 375
4	uvc_ xpcs_ 8hr_ dn uvc_ xpcs_ 4hr_ dn alzheimers_ disease_ up	MALT1 C19ORF7 IFRD1 TCF7L2 FXR1 MYST4 HYPB TSPAN5 PUM1 NCOA3 NFIB ZNF148 CNOT2 RBPSUH FYN RBMS1 CDC2L5 SRPK2 AFF1 PTPRK IGF1R E2F5 KLF7 TGFB2 JUN WRN QKI C13ORF24 C4ORF8 CENTB2 TIPARP CBLB CHD1 C6ORF111 LDLR CUTL1 CTBP2 CREBBP ZNF198 ZHX3 RAI17 UVRAG VEGFC MYST3 PPFIA1 RABGAP1 RNGTT NIPBL LMNB1 EGFR HELZ TLE4 MYO10 BRCA1 CAP350 MARK3	46606 46599 6375 42772 46605 7099 244 51298 376 387 51054 7098 51297 242 51272 30330 30128 6282 45334 30595 4468 912 45937 9452 42423 30518 30132 42417 50900 19198 19204 4709 31032 6020 45793 30139 30131 6584 16540 8629 19220 42325 4714 6739 30334 51052 1932 19199 6383 19722 6473 6563 375 7200 6769 45935 18108 8080 9451 18212 398 377 8380 9070 16410 43066 7249 48015 7169 1505 30125 19319 46364 19362 16407 16072 45934 42401 6397 16485 6733 6396 30136 45941 8094 1558 30120 16570 16569 9069 16481 16071 6366 18193 16310 30135 6351 6323 45449 4721 6461 45892 4713 16311 6325 6917 31497 4702 7001 12502 785 4725 16049 43065 6464 6357 6333 6606 51170 6486 43413 9101 9100 17038 6355 15674 8026 6417 50776 9309 19318 51169 30036 42623 7067 30029 6468 6913 16568 16887 16051 8652 17111 6605 6512 6281 6520 30001 15630 6412 6812 43037 6886 4674
5	uvc_ xpcs_ 8hr_ dn uvc_ xpcs_ 4hr_ dn	MALT1 C19ORF7 IFRD1 TCF7L2 FXR1 MYST4 HYPB TSPAN5 PUM1 NCOA3 NFIB ZNF148 CNOT2 RBPSUH FYN RBMS1 CDC2L5 SRPK2 AFF1 PTPRK IGF1R E2F5 KLF7 TGFB2 JUN WRN QKI C13ORF24 C4ORF8 CENTB2 TIPARP CBLB CHD1 C6ORF111 LDLR CUTL1 CTBP2 CREBBP ZNF198 ZHX3 RAI17 UVRAG VEGFC MYST3 PPFIA1 RABGAP1 RNGTT NIPBL LMNB1 EGFR HELZ TLE4 MYO10 BRCA1 CAP350 MARK3 APPBP2 TRIM33 ZFHX1B PHLPPL UBR2 KLF6 LARP5 RASA1 RASSF3 BDNF DOC1 DOCK4 RUNX1 SLC25A12 HEG1 TRIM2 CDK8 VPS13B TIAM1 EDD1 NAV3 ACSL3 ABCC1 MTAP GLRB IRS1 HOXB2 ASXL1 HRB TSC22D2 MICAL3	1952 6375 46599 46606 42772 45616 244 46605 7099 48625 387 376 51298 15822 51054 5237 1942 1943 45767 45428 7098 6020 42633 51297 8192 16248 8200 31023 242 30128 30330 51272 6282 45334 16933 30228 30595 930 4468 912 30169 45937 42423 9452 30518 30132 15459 6984 42417 30522 50900 19198 19204 31032 4709 45793 6809 46209 42136 17145 30139 42770 30131 35050 109 6584 15645 8016 16540 8629 51270 5813 30041 19319 46364 30119 18958 46165 152 48332 51427 35257 5254 45445 19220 51174 42133 48514 16408 8544 1704 51187 40017 42325 4714 1568 6739 30334 31589 30118 45596 5253 48730 7569 42379 375 9891 51052 8064 5815 1932 19199 43066 5905 30832 6383 15807 398 377 19722 43069 3712 6473 6839 42127 6563 8380 51241 30117 45860 51347 8154 82 7169 7200 6769 45935 8361 6778 1664 31325 51258 1709 51242 18108 8080 45934 6521 9451 18212 9893 9070 15276 16410 7249 51240 48015 7160 7519 31324 5230 6397 51243 5231 16591 1505 16477 51028 30125 19362 43067 16407 6396 16072 7167 7369 30029 16747 6928 7507 16071 9892 31323 19219 6935 42401 16485 6733 42113 16070 42981 6793 30036 46930 12501 45449 6351 42330 6954 6366 6916 30136 19318 45941 46914 5996 1558 8094 6796 30098 6350 30120 16570 16569 9069 6066 16481 9611 16310 18193 8415 50658 51236 50657 6576 6405 910 6915 30135 42398 6323 31226 6357 4721 6461 16791 15931 45892 51276 51329 51128 4713 16881 16311 6325 6917 6139 31497 43283 6464 4702 43412 7001 74 9889 6974 5085 12502 30005 42598 785 51168 4672 43068 6575 4725 42578 46483 30554 16049 16051 43065 7243 6875 19932 16773 7268 15630 45859 5635 51246 51169 30003 6333 6259 6873 902 6996 51325 7242 6606 51170 278 6486 19226 87 43413 16301 6913 9101 51186 6355 4197 44262 15698 9100 17038 15293 7010 9613 43231 15674 5125 5654 6865 8026 19538 44267 30097 44260 4527 6767 5694 3779 6417 6820 43207 19941 16023 31410 15837 50776 9309 44271 5975 43232 46907 46942 15849 44248 51649 279 51603 44257 31326 5856 42623 7067 46649 6766 6519 6468 45321 6955 16758 30163 7166 17111 16462 43285 9308 16568 16887 16818 42221 8234 5740 6952 5887 16779 6725 15031 9059 8652 44249 19001 6605 16021 45184 6732 5794 6512 6281 6810 6520 30001 9057 15291 19752 44265 6082 6811 6412 6812 43037 16192 6886 4518 4674 4175 44255 6508 6629 7186 6091 5739 9117

Continued on next page

Concepts	Objects	Attributes	Terms
6	uvc_ xpcs_ 8hr_ dn uvc_ ttd-xpcs_ common_ dn alzheimers_ incipient_ up alzheimers_ disease_ up	MALT1 NUP98 TRIM44 GOLGA4 RAP140	46707 51226 9179 212 51225 7051 9206 9145 6606 9201 9152 9205
7	uvc_ xpcs_ 8hr_ dn uvc_ ttd-xpcs_ common_ dn alzheimers_ disease_ up	MALT1 C19ORF7 NFIB ITSN1 CDC2L5 IGF1R E2F5 KLF7 NUP98 WRN TRIM44 C4ORF8 CENTB2 TIPARP GOLGA4 KIF14 DST CASK RAP140 PPFIA1 SEPT9 CRADD LARP4 ID3 MARK3	46707 6375 6165 46939 244 45103 387 51226 9179 212 9135 51225 9185 6739 16540 8624 7143 9132 6769 7051 19362 19201 6733 4714 43066 7249 398 377 375 51052 19199 8380 16485 7169 226 6397 6917 12502 7126 43065 16071 6396 9206 9145 6606 51170 9142 9201 9150 6163 7017 17038 6164 9152 9144 9259 9260 9141 16310 9165 15630 4713 9199 51169 9205 7067 6461 30029 6913 6310 6605 6464 6366 6886 7001 42623 6468 16887 17111 6351 6355 45449
8	uvc_ xpcs_ 8hr_ dn uvc_ ttd-xpcs_ common_ dn	MALT1 C19ORF7 NFIB ITSN1 CDC2L5 IGF1R E2F5 KLF7 NUP98 WRN TRIM44 C4ORF8 CENTB2 TIPARP GOLGA4 KIF14 DST CASK RAP140 PPFIA1 SEPT9 CRADD LARP4 ID3 MARK3 APPBP2 TRIM33 ZFHX1B PHLPPL UBR2 KLF6 LARP5 RASA1 RASSF3 BDNF DOC1 DOCK4 RUNX1 SLC25A12 HEG1 TRIM2 CDK8 VPS13B TIAM1 EDD1 NAV3 ACSL3 ABCC1 MTAP GLRB IRS1 HOXB2 ASXL1 HRB TSC22D2 MICAL3	46707 1952 6375 6165 244 46939 387 45103 15822 51226 5237 45767 9179 9925 212 8200 16248 6020 16933 9135 15459 51225 9185 6739 15645 16540 8624 30055 7143 30041 16323 9132 5254 16408 46930 31589 6769 5253 19319 7051 46364 7569 42379 46165 8064 30832 43066 15807 43069 6839 6997 45860 5813 51347 8154 82 398 19362 377 375 1664 51258 19201 5635 8380 6733 4714 15276 7249 7160 7169 5230 5231 51028 51052 30029 43067 48514 19199 16776 6397 16485 42113 42981 1568 6916 7167 30098 12501 16071 19205 3712 50658 51236 50657 51187 6396 6405 5815 45786 30036 226 7010 15931 15630 51329 51128 16591 16831 19318 6917 6915 5996 51243 5085 12502 42598 30005 7126 51327 51321 51168 42127 43068 43065 7243 30054 6875 16070 46914 45859 51169 45934 30003 9206 9145 6066 6873 51325 6767 6606 51170 31324 51242 8361 6778 16881 9142 6913 9201 4197 15698 9150 6163 7017 279 17038 16051 6996 15293 5856 9892 31226 6164 5125 16310 9152 9117 30554 6865 6766 9144 30097 4527 9259 6139 278 46907 6820 9260 9141 5794 51649 9165 19941 15837 6796 19219 4713 9199 46942 15849 6793 43283 45449 16773 4672 31323 6351 44262 9205 74 51603 44257 7067 6461 5654 6366 6350 6464 46649 87 6732 16301 45321 16758 30163 45184 43285 51186 902 8234 43412 5740 6357 5975 51246 43232 43231 6310 15031 19001 6605 46483 7166 44260 9057 19538 15291 6810 44267 6259 44265 16021 16791 8415 44249 16747 7242 17111 44248 16462 16818 42578 6886 7001 4518 51276 42623 4175 6468 31410 16887 42221 6955 6508 6355 9059 6952 6091 6811 44255 6629 19752 6082 5739
9	uvc_ xpcs_ 8hr_ dn alzheimers_ incipient_ up alzheimers_ disease_ up	MALT1 NCOA3 RBPSUH FYN SRPK2 EPS8 NUP98 QKI MARCH6 TRIM44 C13ORF24 GOLGA4 ZNF198 ZHX3 LAMC1 ATP8B1 RAP140 RNGTT GSK3B NEK7 SEC14L1 MYO10 CAP350	46707 6165 46939 51226 9179 212 9135 9452 51225 9185 19204 30518 7143 9132 15718 4468 7051 6739 7249 6112 6769 43066 226 19362 7126 6073 8080 9451 9206 9145 6733 16410 6461 6606 51170 16310 45941 7169 45935 9142 9201 16407 16072 9150 6163 7017 17038 6164 15674 9152 6397 6396 9144 16311 9259 398 377 9260 9141 375 9165 16481 16071 9199 46942 51169 45934 8380 9205 6464 6913 4721 6605 30001 4713 19318 45449 6351 6812 6886 42623 4674 16887 6355 17111
10	uvc_ xpcs_ 8hr_ dn alzheimers_ disease_ up	MALT1 C19ORF7 IFRD1 TCF7L2 FXR1 MYST4 HYPB TSPAN5 PUM1 NCOA3 NFIB ITSN1 ZNF148 CNOT2 RBPSUH FYN RBMS1 CDC2L5 SRPK2 AFF1 EPS8 PTPRK IGF1R E2F5 KLF7 NUP98 TGFB2 JUN WRN QKI MARCH6 TRIM44 C13ORF24 C4ORF8 CENTB2 TIPARP CBLB CHD1 C6ORF111 LDLR CUTL1 CTBP2 GOLGA4 CREBBP ZNF198 ZHX3 KIF14 RAI17 LAMC1 ATP8B1 UVRAG VEGFC DST CASK MYST3 RAP140 PPFIA1 RABGAP1 RNGTT SEPT9 NIPBL CRADD GSK3B LMNB1 EGFR NEK7 HELZ TLE4 SEC14L1 MYO10 BRCA1 CAP350 LARP4 ID3 MARK3	46707 6375 46606 46599 42772 45616 6165 46939 46605 7099 48625 244 45103 376 51298 387 51226 51054 1942 1943 45428 7098 42633 51297 8192 9179 9925 212 31023 242 51272 30330 30128 6282 45334 30228 30595 930 4468 30169 45937 912 9135 9452 42423 30518 30132 6984 51225 42417 30522 9185 50900 19198 19204 31032 4709 45793 6020 46209 6809 15247 6739 42136 30139 17145 42770 30131 35050 6584 109 8016 8629 16540 8624 51270 30055 7143 16323 30119 18958 6769 152 51427 48332 35257 9132 45445 51174 19220 51052 42133 15718 8544 1704 51187 40017 42325 4714 30334 5605 30118 45596 7051 48730 3712 19362 9891 1932 48514 19199 5905 6383 19722 6473 42127 6563 6733 6997 1568 51241 30117 46930 45860 5813 51347 51242 375 7169 7200 45935 6778 31325 1709 18108 8080 19201 6521 9451 18212 398 377 5815 8380 9893 9070 16410 43066 7249 51240 43069 48015 31589 7519 43067 1505 8361 16477 30125 6917 19319 46364 12502 16407 16072 7167 46165 6112 43068 7369 51243 43065 6928 7507 6793 16776 5635 45934 6935 6796 12501 42401 6397 16747 16485 16310 42981 42330 6461 6606 6954 51170 6396 30136 31324 45941 8094 1558 31323 30120 16570 16569 9069 16481 16071 19205 9611 6366 19219 18193 16070 6576 51276 910 6915 17038 74 30135 6351 42398 45786 6323 9892 226 45449 7001 4721 45892 8415 4672 4713 16591 16831 6350 30554 16311 6767 6325 31497 6996 6464 4702 43412 9889 6139 6974 43283 785 7126 16773 51327 51321 6073 6259 16791 6575 4725 46483 15662 46914 16049 7243 30054 19932 31226 7268 6357 45859 51169 279 6333 9206 9145 7010 16301 7242 42578 51186 6486 19226 87 30029 43413 5794 6766 9142 6913 9101 15630 9201 19318 4197 6066 9150 9100 6163 7017 6355 5996 9613 6164 31410 9117 15674 9152 51246 8026 9144 4527 44267 5694 9259 42623 3779 6605 278 44260 43231 19538 6417 9260 9141 6732 902 15405 43207 9165 16023 7166 50776 9309 44271 9199 46907 46942 15849 16887 51649 45184 43232 9205 31326 5856 30036 7067 5654 6519 6468 16881 15031 17111 44248 16462 9308 16568 16818 44262 8234 16051 5887 44249 16192 6886 16779 6310 44264 6725 8652 6955 5975 4674 5976 6512 6281 6520 30001 6810 16021 6952 42221 16820 15980 6811 9059 6412 6812 43037 19752 6082 4518 4175 45045 46903 7186 6091 44255 6629

Continued on next page

Concepts	Objects	Attributes	Terms
11	uvc_ xpcs_ 8hr_ dn	MALT1 C19ORF7 IFRD1 TCF7L2 FXR1 MYST4 HYPB TSPAN5 PUM1 NCOA3 NFIB ITSN1 ZNF148 CNOT2 RBPSUH FYN RBMS1 CDC2L5 SRPK2 AFF1 EPS8 PTPRK IGF1R E2F5 KLF7 NUP98 TGFB2 JUN WRN QKI MARCH6 TRIM44 C13ORF24 C4ORF8 CENTB2 TIPARP CBLB CHD1 C6ORF111 LDLR CUTL1 CTBP2 GOLGA4 CREBBP ZNF198 ZHX3 KIF14 RAI17 LAMC1 ATP8B1 UVRAG VEGFC DST CASK MYST3 RAP140 PPFIA1 RABGAP1 RNGTT SEPT9 NIPBL CRADD GSK3B LMNB1 EGFR NEK7 HELZ TLE4 SEC14L1 MYO10 BRCA1 CAP350 LARP4 ID3 MARK3 APPBP2 TRIM33 ZFHX1B PHLPPL UBR2 KLF6 LARP5 RASA1 RASSF3 BDNF DOC1 DOCK4 RUNX1 SLC25A12 HEG1 TRIM2 CDK8 VPS13B TIAM1 EDD1 NAV3 ACSL3 ABCC1 MTAP GLRB IRS1 HOXB2 ASXL1 HRB TSC22D2 MICAL3	1952 46707 6375 46599 46606 45616 42772 6165 244 46939 46605 7099 51341 48625 387 45103 15822 376 51298 51054 51226 5237 1942 1943 45767 45428 7098 6020 42633 51297 8192 9179 9925 8200 16248 212 5066 31023 242 30330 30128 51272 6282 43256 45334 16933 30228 930 30216 30595 4468 30169 45937 912 9135 35051 9452 42423 30518 15459 30132 45178 6984 51225 42417 30522 9185 50900 19198 19204 4709 31032 45793 16247 46209 6809 15247 15457 30155 6739 8034 42136 17145 30139 42770 30131 35050 6584 109 15645 8016 8629 16540 51270 8624 30055 7143 5813 30041 16323 19319 46364 30119 18958 6769 46165 152 51427 35257 48332 9132 5254 51174 45445 19220 51052 42133 15718 48514 16408 8544 1704 51187 40012 40017 48520 42325 4714 5548 1568 30334 48729 46930 1944 31589 5605 30118 50795 45596 5253 42692 7051 48730 3712 7569 19362 42379 375 19058 48646 9891 8134 8064 7169 5815 1932 19199 30832 43066 5905 6383 43005 51336 15807 398 377 19722 43069 42303 6473 6839 42127 6563 8380 6733 6997 51241 30117 48475 45860 51347 51242 8154 82 7398 9888 30674 7200 45935 8361 6778 43119 1664 31325 48522 1709 51258 18108 8080 16563 5635 19201 30880 45934 6521 9451 43067 18212 7167 9893 9070 15276 16410 51243 48523 48518 51093 19904 5083 7249 51240 48015 7160 7519 31324 30029 5230 6397 40011 5231 16591 43118 19838 5604 1505 16477 7611 51028 5102 30125 16251 6917 48519 30695 16265 42981 50794 12502 51244 5319 16407 6396 16072 12501 6112 8219 43068 7369 50791 6793 31214 8639 16747 43065 6928 51674 7507 8283 16071 9892 16877 31323 6796 19219 16776 16310 16746 19318 46983 6935 8015 19222 42401 16485 43235 42113 3678 16070 5996 46914 8092 30036 45449 7568 6351 42330 6461 6606 6954 51170 6366 6916 7626 30136 45165 7498 35295 45941 43085 42802 50790 8094 1558 6066 30098 6915 6350 30120 16570 45595 16569 9069 16481 19205 9611 31226 18193 15630 43169 51301 8415 50658 51236 50657 6576 7010 46849 51169 6405 7049 51276 910 30554 5057 6139 17038 74 6996 30135 42398 43167 45786 6323 226 6357 6464 51338 4672 7001 4721 16791 3702 15931 45892 43412 43283 8047 51329 51128 4713 16831 16773 16881 46872 6403 16311 6767 6325 17076 5515 31497 6913 4702 48534 9889 6974 5085 30005 42598 785 7126 51327 9719 4386 51321 5886 51239 6073 6259 51168 50875 19899 267 16772 6575 4725 42578 46483 15662 44238 16049 8509 5578 16051 7243 30054 6875 50793 7517 5096 19932 7268 16301 44237 46942 15849 7155 45859 3676 16043 1501 51246 30003 8152 7154 279 6333 9206 9145 6873 902 6950 51325 7242 51186 16879 3677 16740 278 6486 19226 48513 166 87 43413 5794 6766 44262 16830 9605 19725 9142 43170 5856 50801 9101 46907 5941 51649 15268 9201 43231 15267 43227 6355 51641 4197 43229 15698 9653 9150 42592 9100 42995 16564 44260 6163 7017 44267 19538 9887 30154 5622 15293 5216 9613 6164 31410 9117 15674 5125 5654 9152 151 7165 43232 43228 6865 8026 5975 31224 9144 30097 4527 5694 9259 7166 48731 51179 42623 3779 6605 30182 6417 6820 45184 9260 9141 42165 6732 7267 15405 43207 9165 42625 51234 8104 19941 16023 16020 15837 50776 9309 44271 31090 50896 15031 9199 16887 17111 16874 44248 16462 16788 9205 16818 9607 44257 51603 50874 31326 15075 7067 9792 9058 46649 45927 44249 6519 6468 45321 6955 16758 6810 30163 16817 6807 16021 16614 43285 9308 16568 42221 8234 15399 9056 5624 5740 4888 6952 5887 40008 12505 7399 16192 6886 16757 16779 5634 6310 44264 6725 9059 8652 4872 19001 4674 19866 9790 6811 5976 1775 6512 6281 6520 30001 16787 9057 42626 43492 15291 15290 5386 16820 19752 9628 44265 6082 16829 5737 15980 48468 6412 6812 43037 8324 4518 6091 3723 50877 4175 45045 44255 6508 46903 6629 16491 7186 8233 30529 5739

Continued on next page

Concepts	Objects	Attributes	Terms
12	uvc_ xpcs_ 4hr_ dn uvc_ ttd-xpcs_ common_ dn alzheimers_ incipient_ up alzheimers_ disease_ up	MALT1 PHF20 THOC2	51028 6405 377 375
13	uvc_ xpcs_ 4hr_ dn uvc_ ttd-xpcs_ common_ dn alzheimers_ disease_ up	MALT1 C19ORF7 PHF20 NFIB CDC2L5 IGF1R E2F5 KLF7 THRAP1 WRN THOC2 C4ORF8 CENTB2 BAZ1A SEC24A TIPARP UGCG BAZ2B PPFIA1 MARK3	6375 6165 6678 46939 46476 46031 244 6677 387 46513 9179 19732 46520 9247 6672 6664 46519 9135 9185 30148 6665 50832 6687 6960 16065 30518 16540 9132 30138 6739 19730 35251 46467 6769 377 375 4714 43066 7249 51028 8380 398 19362 19199 16485 6733 7169 30120 6397 50658 50657 6352 6405 48193 16071 6396 45893 51168 6753 6754 46034 9150 9206 9145 9205 30135 9144 6163 45941 45935 9142 9201 9141 9199 9259 6164 6366 9152 6752 9260 9165 4713 51169 9108 7067 16310 6351 6913 6357 45449 6464 6355 42623 6468 16887 17111
14	uvc_ xpcs_ 4hr_ dn uvc_ ttd-xpcs_ common_ dn	MALT1 C19ORF7 PHF20 NFIB CDC2L5 IGF1R E2F5 KLF7 THRAP1 WRN THOC2 C4ORF8 CENTB2 BAZ1A SEC24A TIPARP UGCG BAZ2B PPFIA1 MARK3 APPBP2 TRIM33 ZFHX1B PHLPPL UBR2 KLF6 LARP5 RASA1 RASSF3 BDNF DOC1 DOCK4 RUNX1 SLC25A12 HEG1 TRIM2 CDK8 VPS13B TIAM1 EDD1 NAV3 ACSL3 ABCC1 MTAP GLRB IRS1 HOXB2 ASXL1 HRB TSC22D2 MICAL3	1952 6375 6165 6678 244 46939 46476 46031 387 6677 15822 5237 45767 46513 9179 19732 46520 16248 8200 9247 6020 6672 16933 6664 46519 9135 15459 9185 30148 6665 50832 6687 6960 16065 30518 15645 16540 30522 30041 51427 35257 9132 5254 16408 30138 9620 6739 31589 51028 5253 19319 46364 7569 19730 42379 46165 35251 8064 43066 30832 15807 377 375 43069 6839 46467 45860 5813 51347 8154 82 398 6643 6769 8380 1664 51258 50658 51236 50657 46527 6405 4714 15276 7249 7160 15931 7169 5230 5231 16591 19362 6397 51168 48514 19199 6959 6733 16485 42113 1568 46930 16071 6396 6916 30029 7167 30098 30120 3712 3713 51187 6352 48193 43067 5815 42981 30036 44255 51329 51128 19318 16070 5996 12501 45893 5085 30005 42598 46914 6629 42127 6754 6753 46034 7243 6875 8610 45859 5635 51169 51243 45934 9150 30003 9206 9145 6915 6066 9205 6873 30135 9144 51325 6163 5654 6366 31324 45941 45935 6351 8361 6778 16758 16881 9142 6913 8194 45449 6139 19219 15630 9201 9141 9199 9259 9117 4197 43207 15698 6350 30117 16051 31323 15293 9892 31226 6357 9613 6164 5125 9152 46907 6865 31325 51649 51242 6752 30097 4527 6767 51186 278 6820 9260 6732 16310 9165 19941 7010 15837 4713 46942 15849 30554 43283 44262 9893 5667 44257 51603 9108 7067 6796 16773 46649 87 4672 6766 45321 6793 30163 43285 6464 902 43231 8234 5740 51188 5856 16301 16192 5975 51246 43412 6810 7242 6952 6955 19001 16462 5794 16818 46483 16023 31410 45045 9057 15291 279 44249 74 44265 44260 16791 46903 8415 6996 19538 6355 16747 15031 44267 7166 17111 44248 45184 43232 42578 4518 16021 42623 4175 6468 16887 42221 6508 9059 6091 6811 6259 19752 6082 5739
15	uvc_ xpcs_ 4hr_ dn alzheimers_ incipient_ up alzheimers_ disease_ up	MALT1 PHF20 NCOA3 CENTD1 RBPSUH FYN SRPK2 QKI THOC2 C13ORF24 AKAP9 ZNF198 ZHX3 TGFBR3 RNGTT MYO10 CAP350	9452 19204 6020 30518 4468 7249 51028 19319 46364 7178 50658 50657 6405 43066 377 375 51168 8080 9451 6397 8380 16410 6396 45941 45935 16071 16407 16072 15674 16311 398 16310 16481 19318 51169 45934 6461 6913 16051 4721 6464 45449 30001 15630 4713 6351 6812 6355
16	uvc_ xpcs_ 4hr_ dn alzheimers_ disease_ up	MALT1 C19ORF7 IFRD1 TCF7L2 FXR1 MYST4 HYPB PHF20 TSPAN5 PUM1 NCOA3 NFIB CENTD1 ZNF148 CNOT2 RBPSUH FYN RBMS1 CDC2L5 SRPK2 AFF1 PTPRK IGF1R E2F5 KLF7 TGFB2 THRAP1 JUN WRN QKI THOC2 C13ORF24 C4ORF8 CENTB2 BAZ1A SEC24A TIPARP CBLB AKAP9 CHD1 C6ORF111 UGCG BAZ2B LDLR CUTL1 CTBP2 CREBBP ZNF198 ZHX3 RAI17 UVRAG VEGFC MYST3 PPFIA1 TGFBR3 RABGAP1 RNGTT NIPBL LMNB1 EGFR HELZ TLE4 MYO10 BRCA1 CAP350 MARK3	6375 46599 46606 45616 42772 6165 6678 46939 46605 46476 7099 244 46031 48625 6677 51298 376 387 51054 1942 1943 45428 7098 42633 51297 46513 8192 9179 19732 46520 30518 31023 242 9247 6020 51272 30128 30330 30522 6282 45334 6672 30228 6664 930 30595 46519 4468 45937 30169 912 9135 9452 42423 30132 6984 42417 9185 50900 19198 19204 51427 35257 4709 31032 45793 30148 46209 6809 6665 42136 50832 6687 6960 30139 17145 42770 16065 30131 35050 6584 109 8016 8629 16540 51270 30119 18958 152 48332 9132 45445 19220 51174 42133 8544 1704 51187 40017 30138 42325 9620 4714 6739 30334 30118 45596 19319 46364 48730 3712 19730 46165 30117 35251 9891 51052 5815 45935 1932 48514 19199 5905 6383 375 19722 31325 6473 42127 6563 1568 46467 51241 45860 5813 51347 377 7200 6643 6769 51242 8380 6778 9893 30120 1709 18108 8080 46527 6521 9451 18212 398 9070 16410 43066 7249 43069 51240 48015 31589 7519 45941 7169 16591 1505 8361 16477 51028 30125 30135 19362 16407 16072 7167 6397 7369 6928 7507 6959 6396 6366 51243 6935 45934 7178 42401 16747 16485 6733 43067 31323 6793 16071 19219 42330 6954 6351 30136 31324 16070 1558 8094 6796 45449 16570 16569 9069 16481 9611 16310 18193 3713 6350 50658 51236 50657 12501 6352 6576 6405 48193 910 42398 42981 6323 9892 6357 9613 6139 4721 6461 15931 45892 8415 44255 43207 51276 4713 16311 46914 6325 6917 31497 6915 4702 7001 74 9889 45893 6974 12502 785 6629 16791 51168 43068 43283 6575 6754 6753 4725 46483 16049 43065 46034 7243 51186 6066 8610 43412 4672 19932 7268 6464 5996 7242 45859 19318 51169 9150 6333 9206 9145 6259 9205 9144 16773 42578 6163 6606 51170 5654 6486 19226 30554 87 31226 43413 9142 16023 31410 6913 9101 6355 8194 9201 9141 9199 9259 9117 4197 9100 16301 17038 16051 6996 6164 15674 9152 51246 8026 6752 4527 43231 6767 5694 3779 278 6417 46907 9260 6732 5794 51649 902 9165 50776 9309 44271 44267 15630 16192 19538 44260 279 6955 44262 5667 31326 30036 9108 42623 6952 7067 43232 30029 6766 6519 6468 16758 16881 44249 7166 44248 16462 9308 16568 16887 16818 8234 5887 51188 5975 16779 6725 15031 8652 17111 6605 45184 6512 6281 6810 45045 6520 30001 9059 42221 5856 46903 6412 6812 43037 19752 6082 6886 4518 16021 4674 4175 6811 7010 7186

Continued on next page

Concepts	Objects	Attributes	Terms
17	uvc_ xpcs_ 4hr_ dn	MALT1 C19ORF7 IFRD1 TCF7L2 FXR1 MYST4 HYPB PHF20 TSPAN5 PUM1 NCOA3 NFIB CENTD1 ZNF148 CNOT2 RBPSUH FYN RBMS1 CDC2L5 SRPK2 AFF1 PTPRK IGF1R E2F5 KLF7 TGFB2 THRAP1 JUN WRN QKI THOC2 C13ORF24 C4ORF8 CENTB2 BAZ1A SEC24A TIPARP CBLB AKAP9 CHD1 C6ORF111 UGCG BAZ2B LDLR CUTL1 CTBP2 CREBBP ZNF198 ZHX3 RAI17 UVRAG VEGFC MYST3 PPFIA1 TGFBR3 RABGAP1 RNGTT NIPBL LMNB1 EGFR HELZ TLE4 MYO10 BRCA1 CAP350 MARK3 APPBP2 TRIM33 ZFHX1B PHLPPL UBR2 KLF6 LARP5 RASA1 RASSF3 BDNF DOC1 DOCK4 RUNX1 SLC25A12 HEG1 TRIM2 CDK8 VPS13B TIAM1 EDD1 NAV3 ACSL3 ABCC1 MTAP GLRB IRS1 HOXB2 ASXL1 HRB TSC22D2 MICAL3	1952 6375 46599 46606 45616 42772 6165 6678 244 46476 46605 46939 7099 46031 48625 51341 387 15822 51298 6677 376 51054 5237 6020 1942 1943 45767 45428 7098 42633 51297 46513 9179 8192 19732 46520 30518 16248 8200 5066 31023 242 9247 30330 30128 51272 30522 6282 45334 6672 16933 30228 6664 30216 930 30595 46519 4468 30169 45937 912 35051 9135 9452 42423 30132 15459 6984 42417 9185 50900 19198 19204 35257 51427 4709 31032 45793 16247 30148 46209 6809 15457 30155 8034 6665 42136 50832 6687 6960 30139 17145 42770 19319 46364 16065 30131 46165 35050 109 6584 15645 8016 8629 16540 51270 5813 30041 30119 18958 152 48332 9132 5254 19220 45445 51174 42133 16408 48514 8544 1704 51187 40012 51336 5048 40017 48520 30138 42325 5815 9620 4714 1568 6739 30334 48729 1944 31589 375 30118 50795 45596 51028 8134 5253 16251 42692 48730 3712 7398 7569 377 19730 42379 19058 48646 48475 30117 35251 9891 51052 8064 8380 45935 1932 19199 30832 5905 43066 43005 6383 15807 398 19722 43069 42303 31325 6473 6839 42127 6563 16591 46467 51241 16563 45860 51347 8154 82 7169 9888 7200 6643 6769 51242 8361 6778 9893 43119 1664 30120 51258 1709 18108 8080 50658 51236 50657 46527 48522 30880 6405 45934 6521 9451 18212 6397 7167 9070 15276 16410 51093 19904 48518 7249 51240 7160 48015 15931 7519 31324 45941 5230 48523 40011 51243 5231 6396 19838 1505 16477 7611 5102 6403 30125 43118 30695 30135 16071 50794 51244 19362 43067 31323 16407 48519 16072 50791 19219 19318 51168 6366 5996 7369 31214 6351 30029 19222 8639 16747 6928 51674 7507 45449 8283 9892 16877 16070 6959 5083 46914 3702 6066 16746 46983 50790 6935 16265 8015 7178 8047 6793 6350 42401 6733 16485 43235 42113 3678 42981 30036 46930 12501 7568 8219 42330 6954 6916 6796 7626 45165 30136 7498 35295 43085 6357 42802 8094 1558 30247 30098 16310 6139 16570 45595 16569 9069 16481 9611 18193 3713 43169 51301 8415 6352 6576 46849 51169 8092 48193 910 5057 1871 6915 16051 42398 6323 31226 44255 43167 9613 51338 4721 6461 16791 45892 43207 15630 51276 51329 51128 4713 43283 16881 44238 16311 6325 6917 31497 6913 9605 6464 4702 48534 6629 43412 7001 74 9889 45893 6974 46872 50875 5085 30005 42598 12502 5515 785 7049 9719 4386 44237 51239 6950 5654 4672 19899 267 43068 6575 6754 4725 6753 42578 46483 30554 3676 16049 8509 43065 46034 7243 6875 50793 7517 51186 8152 5096 8610 19932 16773 7268 7242 7155 45859 1501 5635 51246 9150 30003 6333 9206 9145 44262 6259 9205 6873 902 9144 17076 6996 51325 6163 7154 6606 16879 3677 51170 278 6486 19226 48513 46907 87 43413 16772 6355 51649 16758 19725 9142 16043 51641 16301 16023 31410 50801 9101 8194 43231 5941 9607 5886 15268 9201 9141 9199 43229 43227 16740 43170 9259 15267 48731 9117 4197 15698 9653 42592 9100 42995 5622 17038 9887 30154 15293 50896 7010 5216 6164 15674 16757 5975 5125 166 9152 151 6865 8026 19538 44267 7165 51179 6752 30097 44260 4527 43228 43232 6767 5694 3779 6955 30182 6417 6820 5856 9260 51234 42165 6732 5794 7267 9058 9165 6952 31224 19941 15837 50776 9309 44271 16192 50874 46942 15849 16874 44248 42277 44249 279 16462 30246 16788 7166 16818 5667 51603 44257 31326 9108 42623 7067 9792 46649 45927 6766 6810 6519 16020 6468 45321 15031 30163 4888 6807 17111 45184 16614 43285 9308 16568 16887 15075 42221 8104 8234 9059 9056 5624 5740 5887 40008 51188 7399 4872 16817 16779 6725 8652 19001 6605 19866 5634 9790 16021 1775 6512 6281 45045 6520 31090 30001 9057 5737 15291 15290 16787 19752 9628 44265 46903 6082 6811 48468 6412 6812 43037 6886 4518 3723 4674 50877 4175 5386 12505 6508 16491 7186 8233 8324 30529 6091 5739

Continued on next page

Concepts	Objects	Attributes	Terms
18	uvc_ ttd-xpcs_ common_ dn alzheimers_ incipient_ up alzheimers_ disease_ up	MALT1 PHF20 NUP98 THOC2 TRIM44 GOLGA4 RAP140	46707 6165 46939 51226 9179 212 9135 51225 9185 7143 9132 7051 7249 51028 50658 50657 6405 43066 226 7126 51168 51169 9206 9145 6606 51170 9142 6913 9201 9150 6163 7017 17038 6164 9152 9144 9259 377 9260 9141 375 9165 9199 8380 9205 6461 6397 6605 16071 6396 6886 16310 45449 6351
19	uvc_ ttd-xpcs_ common_ dn alzheimers_ disease_ up	MALT1 C19ORF7 PHF20 NFIB ITSN1 CDC2L5 IGF1R E2F5 KLF7 NUP98 THRAP1 WRN THOC2 TRIM44 C4ORF8 CENTB2 BAZ1A SEC24A TIPARP UGCG BAZ2B GOLGA4 KIF14 DST CASK RAP140 PPFIA1 SEPT9 CRADD LARP4 ID3 MARK3	46707 6375 6165 46939 6678 46476 244 46031 45103 6677 9179 387 51226 9135 46513 9185 19732 46520 9925 212 9247 6672 6664 46519 51225 9132 30148 6665 50832 6687 6960 16065 30518 6739 16540 8624 30522 30055 7143 16323 35257 51427 30138 9620 6769 7051 19730 35251 6997 46467 45860 46930 51347 19362 6643 46527 19201 377 375 6733 4714 43066 7249 43069 31589 51028 8380 3712 398 51052 19199 6959 16776 5635 9206 9145 16485 7169 9150 43067 9142 6163 30120 6397 9144 19205 3713 9201 50658 51236 50657 51187 6352 6405 48193 9205 42981 45786 226 9259 9117 6164 9141 15931 9152 16071 12501 9199 16591 6396 16831 44255 6917 45893 12502 9260 7126 51327 7167 51321 51168 9165 42127 43068 6753 6754 6629 43065 46034 7243 30054 51242 8610 45859 51169 6915 30135 6767 6606 51170 45941 5794 16070 45935 6778 7010 6913 8194 4197 43207 6139 30117 7017 279 6732 17038 51186 9613 5654 6366 16310 46907 31325 6766 51649 6752 4527 51243 6796 15630 4713 6793 6996 9893 74 5667 9108 7067 6461 6351 30554 87 30029 46914 4672 16758 31410 43283 45184 16773 8234 19219 6350 51188 5856 6357 16301 16192 45449 6310 31323 15031 7242 6605 278 44249 46483 6464 16023 45045 43231 6810 6259 43412 46903 7166 6952 16462 16818 6955 31226 6886 7001 4518 51276 6355 42623 43232 4175 6468 44267 16887 44260 19538 17111 44248 16021
20	uvc_ ttd-xpcs_ common_ dn	MALT1 C19ORF7 PHF20 NFIB ITSN1 CDC2L5 IGF1R E2F5 KLF7 NUP98 THRAP1 WRN THOC2 TRIM44 C4ORF8 CENTB2 BAZ1A SEC24A TIPARP UGCG BAZ2B GOLGA4 KIF14 DST CASK RAP140 PPFIA1 SEPT9 CRADD LARP4 ID3 MARK3 APPBP2 TRIM33 ZFHX1B PHLPPL UBR2 KLF6 LARP5 RASA1 RASSF3 BDNF DOC1 DOCK4 RUNX1 SLC25A12 HEG1 TRIM2 CDK8 VPS13B TIAM1 EDD1 NAV3 ACSL3 ABCC1 MTAP GLRB IRS1 HOXB2 ASXL1 HRB TSC22D2 MICAL3	1952 46707 6375 6165 46939 6678 244 46476 46031 387 45103 15822 6677 9179 51226 5237 45767 9135 46513 9185 19732 46520 9925 212 5066 16248 8200 9247 6020 6672 16933 6664 46519 15459 45178 51225 9132 16247 30148 30155 15457 6665 50832 6687 6960 16065 30518 6739 15645 16540 30522 8624 30055 7143 30041 16323 51427 35257 5254 16408 30138 9620 46930 31589 6769 51028 5253 19319 7051 46364 7569 19730 42379 46165 35251 8064 30832 43066 51336 15807 377 375 43069 6839 6997 46467 45860 5813 51347 8154 82 398 19362 6643 8380 1664 51258 50658 51236 50657 46527 5635 19201 6405 6733 4714 15276 19904 5083 7249 7160 15931 5230 7169 5231 16591 19838 8134 6403 16251 30695 3712 6397 51168 51052 30029 8639 43067 48514 19199 16877 6959 16776 9206 9145 16485 42113 42981 1568 1944 16071 7568 6396 6916 43085 9150 7167 16265 30098 9142 6163 30120 12501 8219 9144 19205 3713 9201 51187 6352 51169 48193 5815 50790 9205 45786 30036 267 226 9259 9117 7010 6164 51338 9141 9152 5102 44255 8047 15630 51329 51128 9199 16831 19318 16563 46914 6917 6913 48523 6915 16070 5996 51243 45893 7398 5085 42598 12502 30005 9260 7126 51327 51321 6629 9165 19899 42127 43068 6753 6754 8509 43065 46034 7243 30054 6875 51242 5096 8610 45859 43169 6139 45934 30003 43118 48519 6066 46907 30135 6873 48522 51325 6767 51649 50794 6606 51244 51170 5654 6366 51641 31324 43167 45941 5794 45935 6351 8361 6778 16758 16830 16881 19725 19219 7155 50801 8194 7049 5941 43119 45449 16310 15268 50791 15267 4197 8092 46872 43207 31323 15698 44238 6350 42592 30117 48475 48518 16564 7017 5057 279 6732 17038 16051 6996 15293 5856 9892 31226 6357 51186 5216 9613 16757 3702 5125 16043 30554 6865 31325 19222 6766 6752 50875 30097 4527 6796 44237 16879 48534 5515 278 6793 17076 6820 42165 4386 5886 51179 43283 19941 8152 15837 31090 8283 4713 46942 15849 51234 16773 4672 45184 44262 9893 74 5667 51603 44257 8104 43227 9108 48731 16772 43231 7067 6461 6464 46649 87 9607 16301 7154 43229 45321 3676 166 15031 30163 31410 6810 5622 51301 9888 9058 43285 15075 902 16874 9605 16740 44249 3677 8234 43412 5578 5624 5740 51188 12505 16192 31224 5975 43170 51246 43228 43232 6310 7242 16020 6952 6955 19001 6605 19866 16462 16818 1775 46483 16023 45045 7166 6950 44260 9057 7165 19538 15291 15290 50896 44267 9056 6259 44265 16021 16791 46903 8415 16829 6355 16747 5634 48513 16817 17111 16746 44248 9719 42578 9653 50874 6886 7001 4518 51276 42623 4175 6468 9887 16788 5737 16787 30154 16887 42221 5386 4888 6508 9059 4872 8233 8324 6091 9628 6811 16491 19752 6082 5739

Continued on next page

Concepts	Objects	Attributes	Terms
21	alzheimers_ incipient_ up alzheimers_ disease_ up	MALT1 PHF20 NCOA3 CENTD1 RBPSUH FYN SRPK2 EPS8 NUP98 QKI THOC2 MARCH6 TRIM44 C13ORF24 AKAP9 GOLGA4 ZNF198 ZHX3 LAMC1 ATP8B1 RAP140 TGFBR3 RNGTT GSK3B NEK7 SEC14L1 MYO10 CAP350	46707 6165 46939 51226 8192 9179 212 9135 9452 51225 9185 19204 6020 15247 30518 30522 7143 35257 51427 9132 15718 4468 5605 7051 6997 46930 45860 51347 6739 7249 51028 19319 46364 46165 6112 5635 7178 6769 50658 51236 50657 51187 6405 5815 43066 226 43069 15931 19362 377 375 7126 51327 7167 51321 6073 51168 15662 8080 45859 51169 9451 6397 9206 9145 8380 6733 16410 6461 6606 51170 6396 16310 45941 7169 45935 6778 9142 6913 6796 16071 3712 9201 16407 6793 16072 19318 4197 9150 6163 7017 17038 16070 5996 6164 15674 9152 31325 16747 43067 9144 9117 16311 9259 398 9260 30554 9141 5794 15405 9165 4672 12501 7243 6066 16481 9199 46942 15849 46914 45934 16773 279 6915 9205 9893 42981 16791 8415 31324 46907 6464 51649 6139 43412 7242 51243 16301 43283 42578 8234 16051 9892 31226 4721 16779 19219 44262 44264 31323 6767 6605 45184 45449 5976 46483 7010 7166 5975 30001 15630 4713 6351 6350 16820 51186 44260 44267 6810 15980 6766 6811 51242 15031 19538 6812 16462 16818 43231 43232 5887 44249 16192 6886 6996 42623 4674 4175 6732 45045 16021 16887 6259 5856 46903 6355 17111 44248 6091 9059
22	alzheimers_ disease_ up	MALT1 C19ORF7 IFRD1 TCF7L2 FXR1 MYST4 HYPB PHF20 TSPAN5 PUM1 NCOA3 NFIB CENTD1 ITSN1 ZNF148 CNOT2 RBPSUH FYN RBMS1 CDC2L5 SRPK2 AFF1 EPS8 PTPRK IGF1R E2F5 KLF7 NUP98 TGFB2 THRAP1 JUN WRN QKI THOC2 MARCH6 TRIM44 C13ORF24 C4ORF8 CENTB2 BAZ1A SEC24A TIPARP CBLB AKAP9 CHD1 C6ORF111 UGCG BAZ2B LDLR CUTL1 CTBP2 GOLGA4 CREBBP ZNF198 ZHX3 KIF14 RAI17 LAMC1 ATP8B1 UVRAG VEGFC DST CASK MYST3 RAP140 PPFIA1 TGFBR3 RABGAP1 RNGTT SEPT9 NIPBL CRADD GSK3B LMNB1 EGFR NEK7 HELZ TLE4 SEC14L1 MYO10 BRCA1 CAP350 LARP4 ID3 MARK3	46707 46599 6375 46606 6165 46939 42772 45616 6678 46605 46476 7099 51341 46031 48625 244 45103 6677 51298 376 9179 387 51054 51226 1942 1943 45428 7098 9135 42633 51297 46513 8192 9185 19732 9925 46520 30518 212 31023 242 9247 6020 30330 30128 51272 30522 6282 43256 45334 6672 30228 6664 30595 30216 930 46519 4468 30169 912 45937 42423 9452 35051 30132 6984 45178 51225 42417 50900 19198 19204 35257 51427 9132 31032 4709 45793 30148 46209 6809 15247 6739 8034 6665 42136 50832 6687 6960 17145 30139 42770 16065 30131 35050 6584 109 8016 8629 16540 51270 8624 30055 7143 16323 30119 18958 6769 152 48332 45445 19220 51174 51052 42133 15718 8544 1704 51187 40012 40017 5048 48520 30138 42325 9620 4714 5548 8134 30334 48729 3712 5605 30118 50795 45596 16251 42692 19319 7051 46364 48730 7398 19362 19730 19058 46165 48646 48475 30117 35251 9891 5815 45935 1932 48514 19199 5905 6383 43005 51336 375 19722 42303 31325 6473 42127 6563 9888 6733 51242 6997 1568 46467 51241 16563 1944 45860 46930 5813 51347 43119 377 7169 7200 30674 6643 48522 8380 6778 9893 30120 1709 18108 8080 46527 19201 30880 6521 9451 48518 18212 398 7167 9070 16410 43066 51093 7249 51240 43069 48015 31589 7519 45941 40011 16591 19838 5604 43067 1505 8361 16477 7611 51028 30125 6917 30135 12502 5319 16407 16072 6397 6112 43068 7369 6793 31214 51243 43065 6928 51674 50794 7507 51244 8283 48523 6959 6796 6396 50791 3702 6366 16776 16310 46983 5635 45934 6935 8015 7178 12501 42401 43118 16747 9206 9145 16485 31323 43235 3678 16265 42981 19219 48519 16071 8219 42330 16746 6606 6461 19222 6954 51170 6351 7626 45165 30136 7498 31324 35295 43085 9150 42802 16070 50790 5102 1558 8094 30247 9142 45449 6163 16570 45595 16569 9069 16481 6139 9144 19205 9611 18193 3713 6350 9201 50658 51236 50657 6352 6576 46849 51169 8092 6405 48193 51276 910 1871 6915 17038 9205 74 42398 45786 6323 9892 9117 226 6357 9259 46914 9613 6164 51338 7001 4721 9141 15931 45892 9152 8415 44255 8047 43207 4672 4713 9199 7049 16831 19318 30554 6403 16311 6767 6325 31497 6913 51186 43283 9605 30695 6996 6464 4702 5996 43412 44238 9889 45893 6974 6066 51301 9260 785 7126 16773 43169 51327 5794 4386 51321 51239 6629 6073 6259 16791 51168 9165 50875 6575 6753 4725 6754 46483 15662 44237 16049 5578 46034 7243 30054 6950 50793 7517 8610 5515 19932 31226 7268 15630 43167 7242 16772 45859 1501 17076 8152 279 6333 7010 31410 9719 16301 5886 46872 42578 7154 5654 6486 19226 46907 87 30029 16043 43413 6732 3676 6766 51649 16830 16740 51641 16023 9101 6355 8194 4197 9100 42995 16564 7017 5057 16051 43231 9607 166 43229 3677 16192 43170 48513 43227 15674 5083 51246 151 5622 50896 8026 7165 6752 4527 44267 51179 5694 7166 42623 3779 9058 48534 6605 278 44260 30182 19538 43228 43232 6417 44262 45184 5856 30154 7267 9653 902 9887 15405 42625 267 51234 8104 44249 50776 9309 44271 15031 46942 15849 16887 42277 6955 16462 30246 16818 5667 31326 30036 9108 48731 6952 7067 5975 31224 9792 45927 6519 50874 6468 16758 16881 16020 6810 4888 16788 45045 16817 7155 6807 17111 44248 16614 9308 16568 8234 15399 5887 40008 51188 7399 6886 16757 46903 16779 5634 6310 44264 6725 8652 4872 16879 4674 9790 5976 6512 6281 6520 31090 5737 30001 9059 16021 42626 43492 42221 16787 16820 12505 16829 15980 6811 48468 6412 6812 43037 19752 6082 4518 9056 9628 3723 50877 4175 16874 5386 7186 8233 8324 15075 30529 6091 16491

Continued on next page

Concepts	Objects	Attributes	Terms
23		MALT1 C19ORF7 IFRD1 TCF7L2 FXR1 MYST4 HYPB PHF20 TSPAN5 PUM1 NCOA3 NFIB CENTD1 ITSN1 ZNF148 CNOT2 RBPSUH FYN RBMS1 CDC2L5 SRPK2 AFF1 EPS8 PTPRK IGF1R E2F5 KLF7 NUP98 TGFB2 THRAP1 JUN WRN QKI THOC2 MARCH6 TRIM44 C13ORF24 C4ORF8 CENTB2 BAZ1A SEC24A TIPARP CBLB AKAP9 CHD1 C6ORF111 UGCG BAZ2B LDLR CUTL1 CTBP2 GOLGA4 CREBBP ZNF198 ZHX3 KIF14 RAI17 LAMC1 ATP8B1 UVRAG VEGFC DST CASK MYST3 RAP140 PPFIA1 TGFBR3 RABGAP1 RNGTT SEPT9 NIPBL CRADD GSK3B LMNB1 EGFR NEK7 HELZ TLE4 SEC14L1 MYO10 BRCA1 CAP350 LARP4 ID3 MARK3 APPBP2 TRIM33 ZFHX1B PHLPPL UBR2 KLF6 LARP5 RASA1 RASSF3 BDNF DOC1 DOCK4 RUNX1 SLC25A12 HEG1 TRIM2 CDK8 VPS13B TIAM1 EDD1 NAV3 ACSL3 ABCC1 MTAP GLRB IRS1 HOXB2 ASXL1 HRB TSC22D2 MICAL3	1952 46707 6375 46599 46606 6165 46939 45616 42772 6678 244 46476 46605 7099 51341 46031 48625 387 45103 376 51298 6677 15822 9179 51226 51054 5237 6020 1942 1943 45767 45428 7098 9135 42633 51297 46513 8192 9185 19732 46520 9925 30518 16248 212 8200 5066 31023 242 9247 30330 30128 51272 30522 6282 43256 45334 6672 16933 30228 6664 930 30216 30595 46519 4468 30169 912 45937 42423 35051 9452 15459 30132 45178 6984 51225 42417 50900 19198 19204 35257 51427 9132 4709 31032 45793 16247 30148 6809 46209 15457 15247 30155 6739 8034 6665 42136 50832 6687 6960 30139 17145 42770 19319 46364 16065 30131 46165 35050 6584 109 15645 8016 16540 8629 8624 51270 30055 7143 5813 30041 16323 30119 18958 6769 152 48332 5254 51174 19220 45445 51052 42133 48514 16408 15718 8544 1704 51187 51336 40012 40017 48520 5048 30138 42325 5815 9620 4714 5548 1568 8134 30334 46930 48729 1944 3712 31589 5605 375 30118 50795 45596 51028 5253 16251 42692 7051 48730 7398 7569 19362 377 19730 42379 19058 48646 48475 30117 35251 9891 8064 7169 8380 45935 1932 19199 43066 5905 30832 43005 6383 15807 398 19722 43069 42303 31325 6839 6473 42127 6563 9888 16591 6733 51242 6997 46467 51241 16563 45860 51347 8154 43119 82 30674 7200 6643 7167 48522 8361 6778 9893 1664 30120 1709 51258 18108 8080 50658 51236 50657 46527 19201 5635 30880 6405 45934 6521 9451 43067 48518 18212 6397 9070 30695 15276 16410 51243 48523 51093 19904 5083 7249 51240 48015 7160 15931 5102 7519 31324 45941 30029 5230 40011 5231 6396 43118 19838 5604 1505 16477 7611 19318 50794 51244 6403 30125 6917 48519 5996 50791 30135 16071 16265 42981 31323 12502 19219 5319 16407 6793 16072 12501 51168 6366 6112 8219 43068 7369 19222 31214 6351 46914 8639 16747 43065 6928 51674 6796 7507 6066 16310 45449 8283 9892 16877 16070 6959 3702 16776 16746 46983 50790 51169 6935 8015 7178 8047 6350 42401 9206 9145 16485 43235 42113 3678 8092 30036 6139 7568 15630 42330 6461 6606 6954 51170 6916 7626 45165 30136 7498 35295 43169 43085 9150 6357 42802 8094 1558 30247 30098 6915 9142 6163 6913 16570 45595 16569 9069 16481 43167 9144 19205 9611 31226 18193 3713 51301 9201 8415 6352 6576 7010 46849 48193 7049 51276 910 30554 5057 1871 17038 16051 9205 74 6996 42398 45786 6323 44238 9117 267 46872 226 44255 9259 43283 6464 9613 6164 51338 4672 7001 4721 9141 16791 5515 9152 45892 43412 50875 43207 51329 51128 4713 9199 16831 44237 16773 16881 16311 6767 6325 17076 31497 51186 9605 4702 48534 6629 9889 8152 45893 6974 5085 42598 30005 46907 9260 785 7126 51327 5794 9719 4386 51321 5886 51239 51649 6073 6950 6259 5654 16043 9165 19899 16772 6575 4725 6754 6753 42578 51641 46483 15662 7154 3676 16049 8509 5578 46034 7243 30054 44262 6875 50793 7517 5096 8610 19932 7268 16301 7242 46942 15849 7155 45859 1501 51246 30003 279 6333 31410 6873 902 5856 51325 16740 16879 3677 43231 278 6486 19226 48513 166 43229 87 43227 43413 6732 6355 6766 43170 16758 16830 19725 16023 51179 5622 50801 9101 8194 5941 9607 5975 15268 43228 43232 15267 48731 51234 4197 15698 9653 42592 9100 42995 16564 44260 7017 44267 19538 7165 9887 30154 15293 50896 16192 5216 45184 15674 7166 16757 9058 5125 151 31224 6865 8104 8026 6752 30097 44249 4527 15031 5694 42623 3779 6955 6605 30182 6417 6820 16462 42165 7267 6810 16818 16020 15405 42625 6952 19941 15837 50776 9309 50874 44271 31090 16887 17111 16874 44248 42277 30246 16788 16817 5667 51603 44257 31326 15075 9108 7067 9792 46649 45927 6519 6468 45321 30163 4888 45045 6807 16021 16614 43285 9308 16568 42221 5634 8234 9059 15399 9056 5624 5740 5887 40008 51188 12505 7399 6886 4872 46903 16779 6310 44264 6725 8652 19001 4674 5737 19866 9790 6811 5976 1775 6512 6281 16787 6520 30001 9057 42626 43492 15291 15290 5386 16820 19752 9628 44265 6082 16829 15980 48468 6412 6812 43037 8324 4518 6091 3723 50877 4175 6508 16491 7186 8233 30529 5739

Bibliography

[1] *Inferno Manual, Fourth Edition (Manual Pages)* . Vita Nuova Ltd.

[2] *Plan 9 Programmers Manual, Fourth Edition (Manual Pages)*. Bell Labs Lucent Technologies, 2002.

[3] Concept Lattices, Second International Conference on Formal Concept Analysis, ICFCA 2004, Sydney, Australia, February 23-26, 2004, Proceedings. In Peter W. Eklund, editor, *ICFCA*, volume 2961 of *Lecture Notes in Computer Science*. Springer, 2004.

[4] S. Aburatani, K. Tashiro, C. J. Savoie, M. Nishizawa, K. Hayashi, Y. Ito, S. Muta, K. Yamamoto, M. Ogawa, A. Enomoto, M. Masaki, S. Watanabe, Y. Maki, Y. Takahashi, Y. Eguchi, Y. Sakaki, and S. Kuhara. Discovery of novel transcription control relationships with gene regulatory networks generated from multiple-disruption full genome expression libraries. *DNA Res*, 10(1):1–8, 2003. 1340-2838 Journal Article.

[5] Alfred V. Aho and Jeffrey D. Ullman. The universality of data retrieval languages. In *POPL*, pages 110–120, 1979.

[6] Hal Alper and Gregory Stephanopoulos. Global transcription machinery engineering: a new approach for improving cellular phenotype. *Metab Eng*, 9(3):258–267, May 2007.

[7] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–402, 1997. 0305-1048 Journal Article Review Review, Tutorial.

[8] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature Genetics*, 25(1):25–9, 2000.

[9] Michael Bada, Robert Stevens, Carole A. Goble, Yolanda Gil, Michael Ashburner, Judith A. Blake, J. Michael Cherry, Midori A. Harris, and Suzanna Lewis. A short study on the success of the Gene Ontology. *Journal Web Semantics*, 1(2):235–240, 2004.

[10] Marcello Barbieri. *Organic Codes: An Introduction to Semantic Biology*. Cambridge University Press, 2003.

[11] S. Beck, A. Olek, and J. Walter. From genomics to epigenomics: a loftier view of life. *Nature Biotechnology*, 17:1144, Dec 1999.

[12] Amir Ben-Dor, Benny Chor, Richard Karp, and Zohar Yakhini. Discovering local structure in gene expression data: the order-preserving submatrix problem. *Journal Computational Biology*, 10(3-4):373–384, 2003.

[13] J. D. Blower, A. B. Harrison, and K. Haines. Styx grid services: Lightweight, easy-to-use middleware for scientific workflows. In Vassil N. Alexandrov, G. Dick van Albada, Peter M. A. Sloot, and Jack Dongarra, editors, *Computational Science - ICCS 2006, 6th International Conference, Reading, UK, May 28-31, 2006, Proceedings, Part III*, volume 3993 of *Lecture Notes in Computer Science*, pages 996–1003. Springer, 2006.

[14] Evelyn Camon, Michele Magrane, Daniel Barrell, Vivian Lee, Emily Dimmer, John Maslen, David Binns, Nicola Harte, Rodrigo Lopez, and Rolf Apweiler. The Gene Ontology Annotation (GOA) Database: sharing knowledge in Uniprot with Gene Ontology. *Nucleic Acids Research*, 32(Database issue):262–266, Jan 2004.

[15] Claudio Carpineto and Giovanni Romano. *Concept Data Analysis: Theory and Applications*. John Wiley & Sons, 2004.

[16] Yizong Cheng and George M. Church. Biclustering of expression data. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 93–103.

[17] Dominique Chu and Weng Kin Ho. A category theoretical argument against the possibility of artificial life: Robert rosen's central proof revisited. *Artificial Life*, 12(1):117–134, 2006.

[18] Dominique Chu and Weng Kin Ho. The localization hypothesis and machines. *Artificial Life*, 13(3):299–302, 2007.

[19] Philipp Cimiano, Andreas Hotho, Gerd Stumme, and Julien Tane. Conceptual knowledge processing with formal concept analysis and ontologies. In *International Conference on Formal Concept Analysis*, pages 189–207, 2004.

[20] A Cornish-Bowden and M L Cardenas. From genome to cellular phenotype–a role for metabolic flux analysis? *Nature Biotechnology*, 18(3):267–268, Mar 2000. Comment.

[21] Athel Cornish-Bowden. Putting the systems back into systems biology. *Perspectives in Biology and Medicine*, 49(4):475–489, Autumn 2006.

[22] Mark A. Sheldon David K. Gifford, Pierre Jouvelot and James W. O'Toole Jr. Semantic file systems. *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 16–25, October 1991.

[23] Anita de Waard, Leen Breure, Joost G. Kircz, and Herre van Oostendorp. Modelling Rhetoric in Scientific Publications. In *INSCIT*, 2006.

[24] J. DeRisi, L. Penland, P. O. Brown, M. L. Bittner, P. S. Meltzer, M. Ray, Y. Chen, Y. A. Su, and J. M. Trent. Use of a cDNA microarray to analyse gene expression patterns in human cancer. *Nature Genetics*, 14(4):457–60, 1996. 1061-4036 Journal Article.

[25] Andreas Doms and Michael Schroeder. GoPubMed: exploring PubMed with the Gene Ontology. *Nucleic Acids Research*, 33(Web Server issue):783–786, Jul 2005.

[26] Sean Dorward and Rob Pike. Programming in limbo. In *Proceedings of the IEEE Compcon 97 Conference*, pages 245–250, San Jose, 1997.

[27] Sorin Draghici, Purvesh Khatri, Rui P Martins, G Charles Ostermeier, and Stephen A Krawetz. Global functional profiling of gene expression. *Genomics*, 81(2):98–104, Feb 2003.

[28] M B Eisen, P T Spellman, P O Brown, and D Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Science U S A*, 95(25):14863–14868, Dec 1998.

[29] D. Engelbart. A conceptual framework for the augmentation of man's intellect. In P. Howerton, editor, *Vistas in Information Handling*, volume 1, pages 1–29. Spartan Books, Washington, DC, 1963. Reprinted in Greif, 1988.

[30] Christiane Fellbaum, editor. *WordNet An Electronic Lexical Database*. MIT Press, 1998.

[31] Sébastien Ferre and Olivier Ridoux. A File System Based on Concept Analysis. In *Computational Logic*, pages 1033–1047, 2000.

[32] Charles H. Forsyth. The Ubiquitous File Server in Plan 9. 2005.

[33] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer Verlag, 1999.

[34] Peter Gärdenfors. *Conceptual Spaces: The Geometry of Thought*. The MIT Press, Cambridge, Massachusetts, 2000.

[35] Audrey P Gasch and Michael B Eisen. Exploring the conditional coregulation of yeast gene expression through fuzzy k-means clustering. *Genome Biology*, 3(11):59, Oct 2002.

[36] M.B. Gerstein, C. Bruce, J.S. Rozowsky, D. Zheng, J. Du, J.O. Korbel, O. Emanuelsson, Z.D. Zhang, S. Weissman, and M. Snyder. What is a gene, post-ENCODE? History and updated definition. *Genome Research*, 17:669–681, Jun 2007.

[37] Dominic Giampaolo. CAT-FS :–a content addressable, typed file system. Master's thesis, Worcester Polytechnic Institute, 1993.

[38] Joseph A. Goguen. What Is a Concept? In *ICCS*, pages 52–77, 2005.

[39] Burra Gopal and Udi Manber. Integrating content-based access mechanisms with hierarchical file systems. In *Proceedings of the 3rd Symposium on Operating Systans Design and Implementation (OSDI-99)*, pages 265–278, Berkeley, CA, February 22–25 1999. Usenix Association.

[40] Eric Grosse. Real Inferno. In Ronald F. Boisvert, editor, *The Quality of Numerical Software: Assessment and Enhancement: Proceedings of the IFIP TC2/WG 2.5 Working Conference on the Quality of Numerical Software, Oxford, United Kingdon, 8–12 July 1996*, pages 270–279, London, 1997. Chapman Hall on behalf of IFIP.

[41] T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[42] Gutiérrez, N. C. and López-Pérez, R. and Hernández, J. M. and Isidro, I. and González, B. and Delgado, M. and Fermiñán, E. and García, J. L. and Vázquez, L. and González, M. and San Miguel, J. F. . Gene expression profile reveals deregulation of genes with relevant functions in the different subclasses of acute myeloid leukemia. *Leukemia*, 19:402–409, Mar 2005.

[43] D. Presotto H. Trickey R. Pike D. Ritchie P. Winterbottom S. Dorward. Inferno^{tm}. In *Proceedings of Compcon 97*, 1997.

[44] Willis Harman. *New Metaphysical Foudations of Modern Science*. 1994.

[45] M W Ho. An exercise in rational taxonomy. *Journal of Theoretical Biology*, 147(1):43–57, Nov 1990.

[46] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, NJ, 1985. & 0-13-153289-8.

[47] Gerard J. Holzmann. Using SPIN. World-Wide Web document, Computing Sciences Research Center, Bell Laboratories, Murray Hill, NJ, USA, 2000.

[48] T Ideker, V Thorsson, J A Ranish, R Christmas, J Buhler, J K Eng, R Bumgarner, D R Goodlett, R Aebersold, and L Hood. Integrated genomic and proteomic analyses of a systematically perturbed metabolic network. *Science*, 292(5518):929–934, May 2001.

[49] Ingvar Johansson, Barry Smith, Katherine Munn, Nikoloz Tsikolia, Kathleen Elsner, Dominikus Ernst, and Dirk Siebert. Functional anatomy: a taxonomic proposal. *Acta Biotheoretica*, 53(3):153–166, 2005.

[50] C. A. Joslyn, S. M. Mniszewski, A. Fulmer, and G. Heaton. The Gene Ontology categorizer. *Bioinformatics*, 20 Suppl 1:I169–I177, 2004.

[51] Cliff A. Joslyn, D. D. G. Gessler, S. E. Schmidt, and K. M. Verspoor. Distributed representations of bio-ontologies for semantic web services. In *The Joint BioLINK and 9th Bio-Ontologies Meeting*, 2006.

[52] Robert E. Kent. The iff foundation ontology. *IJCAI 2001 Ontology Workshop*, 2001.

[53] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison-Wesley, Reading, MA, 1999.

[54] Purvesh Khatri and Sorin Draghici. Ontological analysis of gene expression data: current tools, limitations, and open problems. *Bioinformatics*, 21(18):3587–3595, Sep 2005.

[55] Oliver D King, Rebecca E Foulger, Selina S Dwight, James V White, and Frederick P Roth. Predicting gene function from patterns of annotation. *Genome Research*, 13(5):896–904, May 2003.

[56] R.J. Konopka. Genetics of biological rhythms in Drosophila. *Annual Review of Genetics*, 21:227–236, 1987.

[57] F. William Lawvere and Stephen H. Schanuel. *Conceptual mathematics: a first introduction to categories*. Buffalo Workshop Press, Buffalo, NY, USA, 1991.

[58] S. G. Lee, J. U. Hur, and Y. S. Kim. A graph-theoretic modeling on go space for biological interpretation of gene clusters. *Bioinformatics*, 20(3):381–8, 2004. 1367-4803 (Print) Evaluation Studies Journal Article Validation Studies.

[59] Juan-Carlos Letelier, G. Marín, Jorge Mpodozis, and Jorge Soto Andrade. Anticipatory computing with autopoietic and (m, r) systems. In *HIS*, pages 205–211, 2002.

[60] Albertazzi Liliana. Material and Formal Ontology. pages 199–232, 1996.

[61] G. H. Lincoff. *The Audubon Society Field Guide to North American Mushrooms*. 1981.

[62] C. Lindig. Fast Concept Analysis, 2000.

[63] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[64] J. Liu and W. Wang. Op-cluster: Clustering by tendency in high dimensional space, 2003.

[65] Jinze Liu, Jiong Yang, and Wei Wang. Biclustering in gene expression data by tendency. In *CSB '04: Proceedings of the 2004 IEEE Computational Systems Bioinformatics Conference (CSB'04)*, pages 182–193, Washington, DC, USA, 2004. IEEE Computer Society.

[66] A. H. Louie. A living system must have noncomputable models. *Artificial Life*, 13(3):293–297, 2007.

[67] Sara C Madeira and Arlindo L Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology Bioinformatics*, 1(1):24–45, Jan 2004. Comparative Study.

[68] E.R. Mardis. Anticipating the 1,000 dollar genome. *Genome Biology*, 7:112, 2006.

[69] Ben Martin. Formal Concept Analysis and Semantic File Systems. Springer, 2004.

[70] Ben Martin. The world is a libferris filesystem. *Linux Journal*, 2006(146):7, 2006.

[71] J. P. Massar, Michael Travers, Jeff Elhai, and Jeff Shrager. BioLingua: a programmable knowledge environment for biologists. *Bioinformatics*, 21(2):199–207, 2005.

[72] W K Jr McCoubrey, J F Ewing, and M D Maines. Human heme oxygenase-2: characterization and expression of a full-length cDNA and evidence suggesting that the two HO-2 transcripts may differ by choice of polyadenylation signal. *Arch Biochem Biophys*, 295(1):13–20, May 1992. Comparative Study.

[73] Sun Microsystem. Sun Microsystem. NFS: Network file system protocol specification. RFC1094. 1989.

[74] Peter Morville. *Ambient Findability*. O'Reilly Media, Inc., September 2005.

[75] S. J. Mullender and D. Presotto. Programming Distributed Applications using Plan 9 from Bell Labs. In Babaoglu, editor, *4th European Research Seminar on Advances in Distributed Systems (ERSADS), Bertinoro, Italy*, pages 115–132, Italy, May 2001. Univ. di Bologna.

[76] Sape Mullender and David Presotto. Distributed programming with plan 9. In *European Research Seminar on Advances in Distributed Systems*, 2001.

[77] Shamkant B. Navathe and Upen Patil. Genomic and proteomic databases and applications: A challenge for database technology. In *DASFAA*, pages 1–24, 2004.

[78] J. C. Newman and A. M. Weiner. L2L: a simple tool for discovering the hidden significance in microarray expression data. *Genome Biol*, 6(9):R81, 2005.

[79] J. Nikkila, P. Törönen, Samuel Kaski, J. Venna, E. Castren, and G. Wong. Analysis and visualization of gene expression data using self-organizing maps. *Neural Networks*, 15(8-9):953–966, October-November 2002.

[80] I. Niles and A. Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems*, 2001.

[81] Noble and Denis. Claude bernard, the first systems biologist, and the future of physiology. *Experimental Physiology*, 93(1):16–26, January 2008.

[82] Denis Noble. Modeling the heart–from genes to cells to the whole organ. *Science*, 295(5560):1678–1682, Mar 2002.

[83] Denis Noble. *The Music of Life: Biology beyond the Genome*. Oxford University Press, USA, June 2006.

[84] Peter Øhrstrøm, Jan Andersen, and Henrik Schärfe. What has happened to ontology? In Frithjof Dau, Marie-Laure Mugnier, and Gerd Stumme, editors, *Conceptual Structures: Common Semantics for Sharing Knowledge, 13th International Conference on Conceptual Structures, ICCS 2005, Kassel, Germany, July 17-22, 2005, Proceedings*, volume 3596 of *Lecture Notes in Computer Science*, pages 425–438. Springer, 2005.

[85] Yoann Padioleau, Benjamin Sigonneau, and Olivier Ridoux. LISFS: a logical information system as a file system. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 803–806, New York, NY, USA, 2006. ACM Press.

[86] Rob Pike. Structural Regular Expressions. In *Proceedings of the EUUG Spring 1987 Conference*, pages 21–28, Helsinki, 1987.

[87] Rob Pike. The implementation of newsqueak. *Software - Practice and Experience*, 20(7):649–659, 1990.

[88] Rob Pike. Lexical File Names in Plan 9, or, Getting Dot-Dot Right. In *Proceedings of the 2000 USENIX Technical Conference*, pages 85–92, San Diego, 2000.

[89] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[90] Rob Pike, Dave Presotto, Sean Dorward, Dennis M. Ritchie, Howard Trickey, and Phil Winterbottom. The inferno operating system. *Bell Labs Technical Journal*, 2(1), 1997.

[91] Rob Pike, Dave Presotto, Ken Thompson, and Howard Trickey. Plan 9 from Bell Labs. In *Proceedings of the Summer 1990 UKUUG Conference*, pages 1–9, London, 1990.

[92] Rob Pike, Dave Presotto, Ken Thompson, Howard Trickey, and Phil Winterbottom. The use of name spaces in plan 9. In *Proceedings of the 5th ACM SIGOPS Workshop*, Mont Saint-Michel, 1992.

[93] Rob Pike and Peter Weinberger. The hideous name. In *USENIX Summer 1985 Conference Proceedings*, page 563, Portland Oregon, 1985.

[94] D. Ritchie R. Pike. The Styx architechture for distributed systems. *Bell Labs Technical Journal*, 4(2):146–152, 1999.

[95] N. Ramakrishnan, M. Antoniotti, and B. Mishra. Reconstructing Formal Temporal Models of Cellular Events using the GO Process Ontology. In *Bio-Ontologies SIG Meeting*, 2005.

[96] Ayn Rand. *Introduction to Objectivist Epistemology*. Mentor, 1967.

[97] David J Reiss, Nitin S Baliga, and Richard Bonneau. Integrated biclustering of heterogeneous genome-wide datasets for the inference of global regulatory networks. *BMC Bioinformatics*, 7:280, 2006.

[98] P. Resnik and M. Diab. Measuring verb similarity. In *Twenty-Second Annual Meeting of the Cognitive Science Society*, 2000.

[99] Robert Rosen. *Life Itself*. Columbia University Press, New York, 1991.

[100] M. Satyanarayanan. *Distributed File Systems*. 1993.

[101] M. Schena, D. Shalon, R. W. Davis, and P. O. Brown. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270(5235):467–70, 1995. 0036-8075 Journal Article.

[102] Bernd S. W. Schröder. *Ordered Sets*. Birkhauser Springer.

[103] R Sharan and R Shamir. CLICK: a clustering algorithm with applications to gene expression analysis. *Proceedings of the International Conference on Intelligent Systems for Molecular Biology*, 8:307–316, 2000.

[104] M Sipper and D Mange. Von Neumann's quintessential message: genotype + ribotype = phenotype. *Artificial Life*, 4(3):225–227, Summer 1998.

[105] Barry Smith, Werner Ceusters, Bert Klagges, Jacob Kohler, Anand Kumar, Jane Lomax, Chris Mungall, Fabian Neuhaus, Alan L Rector, and Cornelius Rosse. Relations in biomedical ontologies. *Genome Biology*, 6(5):R46, 2005.

[106] John F. Sowa. Conceptual Graphs for a Data Base Interface. *IBM Journal of Research and Development*, 20(4):336–357, 1976.

[107] B B Stanfield and D D O'Leary. Fetal occipital cortical neurones transplanted to the rostral cortex can extend and maintain a pyramidal tract axon. *Nature*, 313(5998):135–137, Jan 1985.

[108] Philip Stanley-Marbell. *Programming Inferno with Limbo*. Wiley, 2003.

[109] Gerd Stumme, Rafik Taouil, Yves Bastide, Nicolas Pasquier, and Lotfi Lakhal. Computing iceberg concept lattices with Titanic. *Data Knowledge Engineering*, 42(2):189–222, 2002.

[110] Andrew Tanenbaum. *Distributed Operating Systems*. Prentice Hall, Englewood Cliffs, 1 edition, 1995.

[111] Ken Thompson. Reflections on trusting trust. *Commun. ACM*, 27(8):761–763, 1984.

[112] M Travers, JP Massar, and J Shrager. The (re)birth of the knowledge operating system. In *International Lisp Conference*, 2005.

[113] F. J. Varela, H. R. Maturana, and R. Uribe. Autopoiesis: The organization of living systems. *BioSystems*, 5(4):187–196, 1974.
One of the first presentations of the concept of autopoiesis.

[114] Raphael Volz, Siegfried Handschuh, Steffen Staab, Ljiljana Stojanovic, and Nenad Stojanovic. Unveiling the hidden bride: Deep Annotation for Mapping and Migrating Legacy Data to the Semantic Web. *Web Semantics*, 2004. To Appear.

[115] Rudolf Wille. Formal concept analysis as mathematical theory of concepts and concept hierarchies. In *Formal Concept Analysis*, pages 1–33, 2005.

[116] Phil Winterbottom and Rob Pike. The design of the Inferno virtual machine. 1997.

[117] Olaf Wolkenhauer. Interpreting rosen. *Artificial Life*, 13(3):291–292, 2007.

[118] Olaf Wolkenhauer and Jan-Hendrik S. Hofmeyr. An abstract cell model that describes the self-organization of cell function in living systems. *Journal of Theoretical Biology*, 246(3):461–476, June 2007.

[119] Mohammed Javeed Zaki and Naren Ramakrishnan. Reasoning about sets using redescription mining. In Robert Grossman, Roberto Bayardo, and Kristin P. Bennett, editors, *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 364–373. ACM, 2005.

[120] B. R. Zeeberg, H. Qin, S. Narasimhan, M. Sunshine, H. Cao, D. W. Kane, M. Reimers, R. M. Stephens, D. Bryant, S. K. Burt, E. Elnekave, D. M. Hari, T. A. Wynn, C. Cunningham-Rundles, D. M. Stewart, D. Nelson, and J. N. Weinstein. High-Throughput GoMiner, an 'industrial-strength' integrative gene ontology tool for interpretation of multiple-microarray experiments, with application to studies of Common Variable Immune Deficiency (CVID). *BMC Bioinformatics*, 6:168, 2005.

[121] B. Zhang, D. Schmoyer, S. Kirov, and J. Snoddy. GOTree Machine (GOTM): a web-based platform for interpreting sets of interesting genes using Gene Ontology hierarchies. *BMC Bioinformatics*, 5:16, 2004. 1471-2105 Journal Article.