2008

Formal concept analysis and semantic file systems

Ben Martin
University of Wollongong

PhD Thesis: Formal Concept Analysis and Semantic File Systems

by

Mr Ben Martin
B.I.T., Queensland University of Technology
M.I.T., Queensland University of Technology

A thesis submitted to the
School of Information Systems and Technology
University of Wollongong in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
School of Information Systems and Technology
2008

Thesis Certification

CERTIFICATION

I, Benjamin M. Martin, declare that this thesis, submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Information Systems and Technology, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

(Signature)
Benjamin M. Martin
19 October 2008

Ben Martin, Mr (Ph.D., Information Science)

PhD Thesis: Formal Concept Analysis and Semantic File Systems

Thesis directed by Prof. Peter Eklund

The thesis is that a branch of discrete mathematics, Formal Concept Analysis, when applied to Semantic File Systems can lead to an improved personal information space. Semantic File Systems share many properties with their non semantic brethren, bringing more rich metadata and the ability to directly resolve user queries within the filesystem interface itself.

A filesystem might offer upwards of a million files each of which having in the order of hundreds of discerning attributes. Formal Concept Analysis has typically been applied to a much smaller input data set and there are issues with scalability both in the initial finding of the set of Formal Concepts and also ongoing issues such as finding the list of files which are currently applicable (the extent) for a Formal Concept.

The thesis is largely dependent on improving the scalability of Formal Concept Analysis in order for it to be applied to such a large dynamic data store.

Dedication

To the authors of great novels:
Though I have enjoyed many of your works, I have enjoyed too few of your works.

Acknowledgements

Professor Peter Eklund has made this PhD possible. I thank him for his understanding of the value of applied research, the difficulty in performing it and his encouragement and guidance throughout the candidature.

Thanks to associate professor Roger Duke for his guidance during the thesis. His sense of humor brightened the days of many administrative difficulties during the middle of the candidature.

Thanks to Robert Murphy for listening to queries about indexing and relational database design and providing valuable insight into SQL throughout the years both of the PhD and predating it. Apologies for always wanting to talk about libferris over the years.

Contents

Chapter

1 Introduction 1
- 1.1 Hypothesis . . . 2
- 1.2 Prior work on Formal Concept Analysis and File Systems . . . 3
- 1.3 Methodology . . . 5
- 1.4 Major Results . . . 5
- 1.5 Impact and Importance . . . 7
- 1.6 Overall Structure . . . 7

2 Background 9
- 2.1 Introduction . . . 9
- 2.2 Formal Concept Analysis Preliminaries . . . 9
- 2.3 Semantic File Systems . . . 14

3 Indexing Considerations 23
- 3.1 Introduction . . . 23
- 3.2 Indexing full text . . . 24
- 3.3 Indexing metadata . . . 24
  - 3.3.1 Reindexing a File . . . 25
  - 3.3.2 Query Syntax and Semantics . . . 25
  - 3.3.3 Two Designs for the `findex` . . . 26
  - 3.3.4 Specially Sorted Berkeley DB Files . . . 27
  - 3.3.5 Relational Database . . . 29
  - 3.3.6 Index Performance . . . 36
- 3.4 Conclusion . . . 38

4 Formal Concept Analysis and Spatial Indexing 39
- 4.1 Introduction . . . 39
- 4.2 Why Conventional Indexing is Ineffective . . . 40
- 4.3 Spatial Indexing . . . 45
  - 4.3.1 Performance Analysis . . . 50
- 4.4 Asymmetric Page Split Generalized Index Search Trees for Formal Concept Analysis . . . 57
  - 4.4.1 Complete replacement of Guttman . . . 59

4.4.2 Guttman distribution followed by Formal Concept Analysis . . . 60
4.4.3 Customized Key Compression . . . 60
4.4.4 Performance Analysis . . . 64
4.5 Conclusion . . . 72

5 Lattice Closure In A Timely Manner 75
5.1 Introduction . . . 75
5.2 Finding the Closed Frequent Itemsets . . . 77
5.3 A border algorithm . . . 78
5.3.1 An Application Example of the Border Algorithm . . . 79
5.4 A Baseline Algorithm . . . 79
5.5 Performance Analysis . . . 79
5.5.1 Performance on Synthetic data . . . 84
5.5.2 Performance on a Filesystem data . . . 85
5.5.3 Performance on UCI Covtype dataset . . . 87
5.6 Conclusion . . . 87

6 Formal Concept Analysis and Semantic File Systems 89
6.1 Introduction . . . 89
6.2 Application of Formal Concept Analysis . . . 89
6.2.1 Scaling nominal orders . . . 90
6.2.2 Scaling Geospatial information . . . 90
6.2.3 Scaling numeric ranges . . . 91
6.2.4 Natural Groups of Time . . . 93
6.2.5 SELinux . . . 97
6.2.6 Structuring with URLs . . . 100
6.2.7 Context Based Navigation . . . 105
6.3 Conclusion . . . 106

7 System Security and Access Control 111
7.1 Introduction . . . 111
7.2 An Introduction to Access Control . . . 112
7.2.1 Discretionary Access Control . . . 112
7.2.2 Mandatory Access Control . . . 114
7.3 SELinux – DAC and MAC . . . 116
7.4 Prior work on Lattices and Access Control . . . 116
7.5 SELinux and Formal Concept Analysis . . . 120
7.5.1 Holistic Transitive Information Flow . . . 121
7.5.2 Transitive Information Flow from a Fixed Type . . . 121
7.5.3 Single User Access Control . . . 128
7.6 Conclusion . . . 132

8 Advances to Semantic File Systems 138
8.1 Introduction . . . 138
8.2 Supervised Machine Learning and Automatic File Classification . . . 138
8.3 Data models: Semantic Filesystems and XML . . . 143
8.4 Arbitrary Translation Semantic File Systems . . . 146

8.4.1 Relational Models and OpenOffice morphisms 149
8.4.2 Document Computing and The Semantic Web 150
8.5 Conclusion 152
8.6 Semantic File System Future Directions 154

9 Conclusion 155
References 157

Bibliography 157

Tables

Table

3.1 Comparative operators supported by the libferris search syntax. The operators are used infix, there is a key on the left side and a value on the right. The key is used to determine which EA is being searched for. The `lvalue` is the name of the EA being queried. The `rvalue` is the value the user supplied in the query. 26
3.2 Inverted lists are stored in the order of the EA key and EA value. Partial lookups are possible given just the EA key. 28
3.3 Base schema for the `docmap` table. 30
3.4 The `mimetype` table. 30
3.5 Extended `docmap` table. The top of figure is identical to the docmap table from Figure 3.3. 31
3.6 docattrs, the lookup table to document map join table. 31
3.7 Time in seconds to run a query on the `id` EA on an `findex` with the `id` EA normalized or inlined in the `docmap` table. 37
3.8 Benchmarks of running the same base query ($id == 40$) against the many instance `findex` with varying time restrictions. For each benchmark the suitable time restrictions were anded to the base query to limit which instances were considered during `fquery` resolution. 38

5.1 The number of CFI for each configuration. The reduced count is the number of transactions in an object row reduced formal context. The reduced count plays a role in the Covering Edges implementation. As can be seen the reduction process has no bearing for formal contexts with $tlen > 32$. Where the data does not support the requested number of CFI the table has blank cells. 85
5.2 Time taken by various algorithm implementations to make the covering relations between CFI explicit. 86
5.3 Average border size for various CFI data sets. 86
5.4 Performance of intents only and covering edges algorithms on CFI drawn from 100,000 objects with 64 attributes. 88

7.1 Filesystem objects in a DAC system have read, write and execute bits assigned for the owner of the file, those in the owning group and "other" people with no association as either the owner or being in the owning group. 113

7.2 The security context of a running process, the operation requested and the security context of the file determine if an operation will be permitted in SELinux. 115
7.3 The number of incident relations in I for the formal contexts of direct information flows from shadow_t. 125
7.4 The attribute labels for the concept lattice of all security types and attributes which can perform operations on the `shadow_t`. The concept lattice is shown in Figure 7.12. 134

8.1 A medallion is broken down into its individual tags at run-time. The "e:" prefix is a name space prefix which is abridged for presentation purposes. 139

Figures

Figure

2.1 Context of an educational film "Living Beings and Water". 10
2.2 Hasse diagram for the strong groupings for the cross table in Figure 2.1. In Formal Concept Analysis terminology this is the Concept Lattice for the Formal Context shown in Figure 2.1. 12
2.3 The image file Foo.png is shown with it's byte contents displayed from offset zero on the left extending to the right. The png image transducer knows how to find the metadata about the image file's width and height and when called on will extract or infer this information and return it through a metadata interface as an Extended Attribute. 15
2.4 A partial view of a libferris filesystem. Arrows point from children to their parents, file names are shown inside each rectangle. Extended Attributes are not shown in the diagram. The box partially overlapped by `order.xml` is the contents of that file. On the left side, an XML file at path /tmp/order.xml has a filesystem overlaid to allow the hierarchical data inside the XML file to be seen as a virtual filesystem. On the right: Relational data can be accessed as one of the many data sources available though libferris. 18
2.5 The filesystem implementation for an XML file is selected to allow the hierarchical structure of the XML to be exposed as a filesystem. Two different implementations exist at the "order.xml" file level: an implementation using the operating system's kernel IO interface and an implementation which knows how to present a stream of XML data as a filesystem. The XML implementation relies on the kernel IO implementation to provide the XML data itself. 19
2.6 Metadata is presented via the same Extended Attribute (EA) interface. The values presented can be derived from the file itself, derived from the values of other EA, taken from the operating system's Extended Attribute interface or from an external RDF repository. 19
2.7 The three tasks to get from a filesystem to the result of Formal Concept Analysis: the Concept Lattice. 21

3.1 Abstract tuple view of Semantic File System metadata. 23
3.2 A Formal Context for the two term full text query "alice wonderland". 24
3.3 An inverted list with a linear index shown above it. 29

3.4 Core tables in the relational database schema. A single docid in the docmap table can be associated with many tuples in the docattrs table. A single attrid in the attrmap table can be associated with many tuples in the docattrs table. The attrid and docid in the docattrs table can be considered foreign keys. The vid in docattrs can not be a foreign key because it will reference one of many lookup tables (strlookup, intlookup etc) depending on the type of the eavalue that was indexed. 32
3.5 Looking for the `Tokyo.jpg` file by searching for all instances of metadata stored in the `findex` during 2006. 34
3.6 With multiple instances of metadata directly stored in `docmap` and `docattrs` the query must include a subselect to limit consideration to only the most recently added metadata instance for a urlid. 35
3.7 Multiversioned query using negation has undefined semantics. 36

4.1 An inverted file index. For each value of interest there is a list containing all the addresses of tuples which match that value. 41
4.2 Example base relation containing modification and size data for objects. 41
4.3 Ordinal scales on the size, modification and access times of the objects in the base table. Nominal scale on the file-owner. 42
4.4 On the left: B-Tree index on a date column for the base table. Dates in nodes are shown as how long before the current time they represent. The upper nodes are index nodes with the nodes below "12 weeks" omitted. The 17 and 5 days nodes are leaf nodes of the index which point at records in the base table. The B-Tree has a restricted branching factor of two children for illustration purposes. On the right: Resolving the query by a sequential scan filtering out non matching tuples. 43
4.5 Two B-Tree indexes on the base table. The left index is on *modified* while the right index is on *size*. Leaf nodes in both indexes point to tuples physically located throughout the base table. 44
4.6 Expression index on attribute a_1 using f_1, the SQL predicate $size <= 4096$. The B-Tree structure is degenerate because there is only one value indexed. At the leaf page level, pages continue to overflow and the B-Tree approximates in inverted file structure. 46
4.7 An example R-Tree with a query object on the left. Each node has a bounding box which fully contains all objects in its child nodes. An implementation stores the bounding box for each child in the parent node. Note the example is limited to 2 dimensional space with a low branching factor for presentation purposes. 48
4.8 An example RD-Tree with a query object on the left. Each node has a bounding set associated which fully contains all objects in its child nodes. An implementation would store the bounding set for each child in the parent node. Note that the example is limited to only a small set size with a low branching factor in the tree for presentation. 49

4.9 Translating queries involving negation to take advantage of the RD-Tree. This assumes that the attributes 10, 20 and 30 stand for the predicates $a < 10$ and $a < 20$ and $a < 30$ respectively. The weight function returns the number of RD-Tree predicates a tuple contains. So in the above, the third query doesn't need to negate the 30 and 40 predicates because the weight test will already ensure that 30 and 40 are not set. 50
4.10 Selected attributes for the mushroom table and the number of tuples which have the given attribute-value combination. 51
4.11 Times with hot and cold caches to complete queries for 8 attribute list context. Times are in seconds. 52
4.12 Times to complete nested scale queries against the covtype database. The nesting is obtained by generating a nested line diagram in ToscanaJ placing an ordinal scale on elevation inside an ordinal scale on slope. . 52
4.13 Times in seconds with hot and cold caches to complete queries. 53
4.14 Times for query sets against synthetic databases. SQL Explain shows the B-Tree method always electing to disregard all indexes and perform a sequential scan. The RD-Tree query plan always includes zero sequential scans. The number in brackets below the Time column header is the tlen. 54
4.15 Execution times for queries using either B-Tree or RD-Tree indexing against databases of varying density with 10,000 transactions. 54
4.16 Execution times for queries using either B-Tree or RD-Tree indexing against databases of varying density with 1,000,000 transactions. 55
4.17 Statistics for the base table and indexes of the synthetic databases. Note that the B-Tree index size is only for a single column whereas the RD-Tree covers all 32 columns. 55
4.18 Effect of formal context density on RD-Tree performance for 100,000 transaction database. The number of items per pattern was reduced in increments from 128 to 16 giving a max, average and standard deviation of set bits in the formal context as shown. 56
4.19 Basic Generalized Index Search Tree structure. There are four internal nodes (shown at the top) and two leaf nodes (just above the base table) which contain links to the actual information that is indexed. The page size for this tree is illustrative and would normally contain hundreds/thousands of entries. 57
4.20 Pseudo code for asymmetric page split. Preallocation will require a traversal over all keys to be distributed and set union with each key and L and R. The next step is the central part of the algorithm and only loops over keys once. The central distribution will require set unions with each key and both L and R. These can't be cached from the values computed during preallocation because L and R are incrementally expanded during this phase. The final shuffle phase potentially touches most of the keys to be distributed. 61
4.21 Pseudocode for asymmetric page split using guttman and an Formal Concept Analysis postprocess to achieve superior bounding set sizes. 62

4.22 Concept lattice for the source page after Guttman's algorithm has been applied to obtain the initial distribution. The letters above the nodes indicate which attributes are introduced at that concept. When an attribute is introduced at concept x all concepts connected below concept x in the diagram also have that attribute. The numbers below the nodes indicate how many keys match that concept or any connected below it. For example, there is one key with attributes $\{d\}$, four keys with at least $\{b, c\}$ and one with $\{a, b, c\}$. 63
4.23 Compression of a bitset representing a linear scale. 64
4.24 Overall statistics for various Generalized Index Search Tree implementations on the scaled UCI covtype database mediumscaledcov. 66
4.25 Number of keys touched during single attribute extent size queries for the first 128 attributes on an uncompressed index structures. The lowest number to touched keys is always for `Shuffle`. From there upwards are: Asym-NC and RD-NC, in that order. 68
4.26 Number of internal keys touched while querying two attributes against the `RD-Compress` and `Shuffle-Compress` Generalized Index Search Tree for every 16th other attribute with a primary attribute 25. 69
4.27 Mean number of keys touched for single and double attribute queries. . 70
4.28 Overall statistics for various Generalized Index Search Tree implementations on the IBM data mining synthetic database. 70
4.29 Single attribute queries for the first 32 attributes in t100l32n10000plen7i1. This data set involves 100,000 transactions, a total of 10,000 different patterns, a transaction length of 32, a pattern length of 7 items and a total of 1000 different items. 71
4.30 Average number of internal and leaf keys touched for single, two and three attribute queries on various Generalized Index Search Tree implementations. Note that i3 is the internal mean and l3 is the leaf mean. Internal counts are exact, leaf counts are expressed as figures rounded to the nearest hundred. ie. a leaf count in the table of n is for a reading of $n \times 100$ leaf keys. 73

5.1 At the top of the figure is a Formal Context with one of its Formal Attributes and one of its Formal Objects highlighted. The Data Mining perspective is shown below. Formal Objects are seen as Transactions, a group of Formal Attributes is an Itemset. The support for a given Item or Itemset y is the number of transactions which contain a non proper superset that itemset y. 76
5.2 Algorithm to make the order relation between concepts explicit. Input: F the set of concept intents partially ordered on the cardinality of the Intent size from smallest intent size to largest. Output: E an edge mapping from parent concept intent to child concept intent forming the covers for the concept lattice of F. The Intents set introduced on line 8 is also partially ordered from intents with the smallest cardinality to intents with the largest cardinality. 80

5.3 The Maxima function returns the set of intents which are maximal from the given set of intents. The Intents set used as input is ordered from smallest intent cardinality to largest intent cardinality. Line 4 indicates that the ordered Intents poset is to be inspected in reverse order, from largest intent cardinality to smallest. 81
5.4 Steps performed by the border algorithm to find the covers of the concept lattice shown in Figure 5.5. 82
5.5 Concept lattice used as example of border algorithm application. . . . 83
5.6 Modified CoveringEdges using the same syntax as in Concept Data Analysis [20]. As the iceberg lattice does not contain all concepts, the modified version must check that the concept (X_1, Y_1) exists before proceeding. . 83

6.1 Plot of the modification time of 201,759 files from /usr/share/. Horizontal axis shows time from October 1985 to present day with almost 2 years between graduations. Vertical axis ranges from 0 to 3248 with around 235 files separating each graduation. 92
6.2 Plot of the ctime of 201,759 files from /usr/share/. The ctime for a file changes whenever any of its metadata (except atime from lstat) changes. Horizontal axis shows time from 31st January to 05 August 2005 with two and a half weeks between graduations. Vertical axis ranges from 0 to 2494 with around 250 files separating each graduation. 92
6.3 Plot of the ferris-current-time EA of 201,759 files from /usr/share/. . . 93
6.4 Plot of the the width of image files from /usr/share/. 94
6.5 Fewer plot points but a similar overall trend to the width plot. Plot of the megapixels of image files from /usr/share/. 94
6.6 7 formal attributes for each of mtime (modification time) and width using a standard linear range division. Concepts are represented as circles. Labels above a concept show the formal attribute which is introduced by that concept and labels below a concept show the number of filesystem objects which match that concept or one of its refinements. An introduced formal attribute is a formal attribute for which this concept is the highest one in the lattice with that attribute. Thus, where a concept has an introduced formal attribute all concepts reachable transitively downwards will also have this formal attribute. 95
6.7 7 formal attributes for each of mtime (modification time) and width. Formal attributes are generated based on the density of the input metadata. 96
6.8 Numeric group based scaling on time. The concept “1” is selected offering four direct refinements one including the “2” attribute and the other three offering a more restrictive time attribute. 98
6.9 Nominal scaling on time. The concept “1” is selected offering direct refinements to include “2” or a more restrictive time attribute. 98
6.10 Combination of nominal scaling with ordinal scaling applied to group the nominal time attributes. The concept “1” is selected offering direct refinements to include “2” as well as the option of locking the time at Jan06Trip or adding a further restriction to the time attribute to be equal or latter than September 2006. 99

6.11 Concept lattice for SELinux type and identity of files in /usr/share/ on a Fedora Core 4 Linux installation. The Hasse diagram is arranged with three major sections; direct parents of the root are in a row across the top, refinements of SELinux_identity_system_u are down the right side with combinations of the top row in the middle and left of the diagram. 101
6.12 Example lattice with no wordnet augmentation. 102
6.13 Example lattice using wordnet augmentation, notice how the wn_article concept is the common parent of both feature and paper and is also closer to the top of the lattice than either hyponym. 103
6.14 Second example lattice with no wordnet augmentation drawn from an findex of a standard Fedora install. 103
6.15 Example lattice using wordnet augmentation, notice how the wn_article concept is the common parent of both feature and paper and is also closer to the top of the lattice than either hyponym. Unfortunately in this case the "feature" drawn from the redirect.m4 file is a homonym and not the sense that is related to articles. 104
6.16 An iceberg concept lattice showing the 168 Concepts of some geographically tagged digital photographs. The formal context has 92 attributes and 2000 objects. 107
6.17 A context based viewer showing the top node of the iceberg lattice from Figure 6.16. 108
6.18 A context based viewer showing the Germany of the iceberg lattice from Figure 6.16. This figure is obtained by selecting the Germany node in Figure 6.16. There is only the top node as an upper cover and six lower covers. Three of the lower covers are for geographical refinement and have their arrows marked with an "X". Two lower covers are for Time refinements and have their arrow marked with a "T". There is an exposure related refinement marked with an "F". 108
6.19 The iceberg concept lattice from Figure 6.16 centered on the Königsalle in Düsseldorf. There are two time related refinements and a single exposure refinement (marked with a "F" on the arrow). 109
6.20 Navigating the concept lattice from Figure 6.16 as a Semantic File System. The Düsseldorf concept is selected and lower covers are shown in both the file browser on the right and as children in the left tree list. The four virtual directories `all` and `self` allow the user to view the extent or contingent at any concept. The size of the extent of each concept is shown explicitly in the left tree as an EA. 109

7.1 The can-flow relation for a four class Confidentiality information flow policy. Information flows are permitted from the attribute label to any object listed under that label. For example, information in the Secret class can-flow to the Secret and Top-Secret class. 117
7.2 The can-flow function from Figure.7.1 as a Concept Lattice. 118
7.3 A concept lattice for the information flows of two unrelated information classes: medical and payroll data. 118
7.4 Information flow from a file in audio_file_t to an application with type foo_t. 120

7.5 Concept Lattice of the transitive closure of all direct information flows in the SELinux targeted policy version 2.6.4-30.fc7 for Fedora 7. 122
7.6 Formal Concept Analysis of transitive information flow out of the shadow_t security context. The top concept, number 0, has no attributes. Moving down and to the right, concept number 1 has the introduced attributes cardmgr_t. Concept number 2 introduces etc_t and shadow_t, concept 6 introduces var_auth_t and concept 7 introduces security_t. 124
7.7 Concept lattice of the transitive closure of the digraph of degree one or less information flows from shadow_t. The shadow_t attribute is introduced by concept 17. Concept 0 has no attributes. Concept 1 introduces attributes NetworkManager_t and cardmgr_t. Concept 14 introduces attributes nscd_log_t, nscd_t, nscd_var_run_t and samba_log_t. Concept 10 introduces attributes saslauthd_t, saslauthd_tmp_t and saslauthd_var_run_t. 126
7.8 Concept lattice of the transitive closure of the digraph of degree two or less information flows from shadow_t. The shadow_t attribute appears on Concept number 7. Concept 5 introduces attributes automount_t and irqbalance_t. Concept 6 introduces insmod_t and kudzu_t. Concept 4 introduces iptables_t, staff_xserver_t, sysadm_xserver_t, user_xserver_t and xdm_xserver_t. Concept 0 has no attributes. 127
7.9 Concept lattice of standard read, write and execute POSIX protection bits for a sample of files. Three sets of read, write and execute bits exist, one for user, one for group and one for other access. For example, on the far left side of the figure the `/usr/bin/calc` program can be read and executed by everybody on the system. 129
7.10 The concept lattice of the formal context from Table 7.2. It can be easily seen in Table 7.2 that audioplayer_t can be completely overlayed onto bash_t and thus bash_t is a subconcept and there are only the three concepts. 130
7.11 The concept lattice of all security types and attributes which can perform operations on the `shadow_t`. 131
7.12 The concept lattice of all security types and attributes which can perform operations on the `shadow_t`. The attribute labels for each concept are given in Table 7.4. 133
7.13 SELinux policy lines that are used to construct the formal context for the concept lattice in Figure 7.12. The first line means that the `system_chkpwd_t` type can access files of type `shadow_t` if performing a read or a getattr operation. All other types of access to `shadow_t` files for `system_chkpwd_t` will be blocked unless another policy rule explicitly allows it. The last rule uses an SELinux attribute `file_type` to allow `files_unconfined_type` to access all files in general with a broad range of operations. This policy rule affects the `shadow_t` because files of type `shadow_t` will also have the SELinux attribute `file_type` to indicate that they are filesystem files. 135
7.14 The concept lattice of all security types and attributes which can perform operations on the `user_home_t`. 136

7.15 The concept lattice of all security types and attributes which can perform operations on either the `user_home_t` or `shadow_t`. Concept number 2, which is in the middle of the lattice, introduces the shadow_t-read attribute. Concept number 4, which is second from the right in the third row up from the bottom, introduces the shadow_t-append, shadow_t-create and shadow_t-link attributes. Concept 6, which is directly below concept 4, introduces the shadow_t-relabelfrom attribute. Concept 6 also inherits shadow_t-relabelto from Concept 5 which is its other direct cover. Concept 15, which is directly left of concept 6, introduces home_type-write among other attributes. Concept 16 which is the meet of concept 15 and concept 6 and just below them, introduces no attributes and has useradd_t in its extent. Concept 75, the bottom concept, introduces no attributes and has an empty extent. 137

8.1 Viewing the tags associated with file: docs is fully asserted, exe is fully retracted, agents have offered partial retraction on travel and partial assertion on waffle. Assertion is shown in green extending from left to right, retraction is shown in red extending right to left. For readers using a non coloured medium, the tags with ID 10 and 22 are the only green ones. 140

8.2 On the left an XML Element node is shown with some child nodes. On the right a filesystem node is shown with some similar child nodes. Note that XML Text nodes can be considered to provide the byte content of the synonymous filesystem abstraction but metadata about their arrangement can not be easily communicated. Child nodes in the XML side do not need to have unique names for the given parent node and maintain a strict document order. Child nodes on the filesystem side can contain more characters in their file names but the ordering is implementation defined by default. 144

8.3 Union filesystem. The two base filesystems, Base1 and Base2 have their contents combined by set union. Normally there is a linear precedence relation between the base filesystems to explicitly resolve name clashes. In this case Base1's foo.txt will always be selected over the file with the same name in Base2. 146

8.4 A filtering filesystem. Only files matching a given predicate (in this case *.png and *.txt) are exposed from the base filesystem. 147

8.5 A virtual filesystem can be seen equally as a virtual XML document. Notice that the same EAs are shown by both commands, files and directories naturally map to XML entities, EAs map to XML Attributes. 147

8.6 An abstract representation of a schema morphism from a virtual directory to a spreadsheet file. 148

8.7 Schema for msgs table. The id column is the primary key with a default sequential next value if id is not given for a new tuple. 149

8.8 Viewing a mounted PostgreSQL relational database as a virtual filesystem. As can be seen the primary key for this directory is the id metadata. The schema for the msg metadata is a string. 149

8.9 OpenOffice editing a virtual office document created from the current contents of the msgs table in a PostgreSQL database. 151
8.10 Data flow in the creation of a virtual office document from a mounted relational database. 151
8.11 OpenOffice editing a virtual office document created from the current contents of a mounted RDF graph. 152
8.12 Mounting RDF as a filesystem. The RDF graph is stored in a collection of Berkeley db files for fast query processing. These files are named with a foo prefix. 153
8.13 Graphical representation of RDF from Fig 8.12. 153