A Complete Idiot's Introduction to Formal Concept Analysis for Dummies to Teach Themselves

Thomas Sutton

27 November 2013

Is this infringing trade dress infringements. This is satire right?

- The code this talk is about can be found at <https://github.com/thsutton/fca/>.

- It's pretty horrible as of 1/12/2013, but I'll be improving it over the coming weeks.

Caveats

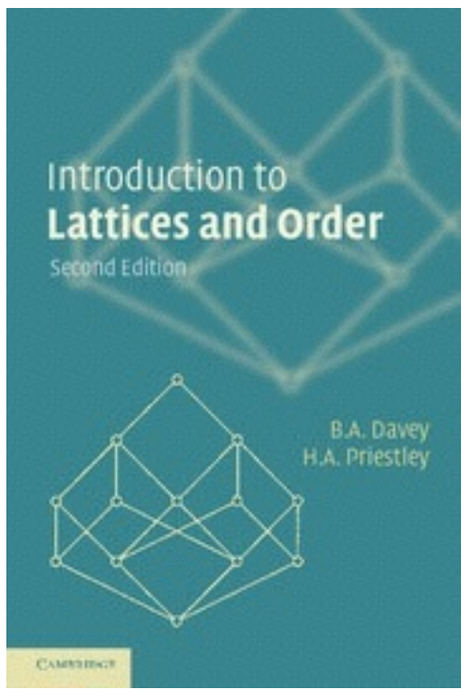
- I’m a pretty bad programmer and this is a talk about some code I wrote.

- I’m pretty bad at mathematics and this talk is me explaining some mathematics.

A long time ago…

- About 10 years ago I visited a branch of the Co-op Bookshop quite regularly and often purchased a book.

- One of them was this:



Alas

"Maths is hard!"

— Me

But!

Chapter 3 is applied.

‡ Sort of.

3

Formal Concept Analysis

Hierarchies occur often both within mathematics and in the 'real' world and the theory of ordered sets and lattices provides a natural setting in which to discuss and analyse them. In this chapter we take a brief excursion into **formal concept analysis** in order to get a feel for the potential of lattice theory in the analysis of hierarchies of concepts.

Contexts and their concepts

3.1 What is a concept? This would appear to be a question for philosophers rather than for mathematicians. Indeed, traditional philosophy's answer provides us with the basis for our formal definition. A concept is considered to be determined by its extent and its intent: the extent consists of all objects belonging to the concept (as the reader belongs to the concept 'living person') while the intent is the collection of all attributes shared by the objects (as all living persons share the attribute 'can breathe'). As it is often difficult to list all the objects belonging to a concept and usually impossible to list all its attributes, it is natural to work within a specific context in which the objects and attributes are fixed.

Formal Concept Analysis

- Formal concept analysis is a mathematical formalism which analyses the data in a **context** and attempts to extract the **concepts** embodied within that data.

- Relating it to completely unrelated techniques for purely intuitive reasons, formal concept analysis might be thought of as the love child of *decision tree learning* and *k-means clustering*.

Context

- A **context** is a structure which relates a set of objects with a set of attributes.

- Formally, a context is a triple:

(G,M,I)

- G (from *gegenstände*) is a set of objects;

- M (from *merkmale*) is a set of attributes; and

- $I \subseteq (G \times M)$ is the relation linking elements of G to elements of M.

Concepts

- A **concept** (with respect to some context) is a pair of sets:

$$(A \subseteq G, B \subseteq M)$$

- A (the *extent*) is the set of all objects which have all the attributes in B; and $\forall a \in G.a \in A \Leftrightarrow (\forall b \in B.b(a))$

- B (the *intent*) is the set of all attributes which apply to all objects in A. $\forall b \in M.b \in B \Leftrightarrow (\forall a \in A.b(a))$

Concepts

- We can derive a concept from *either* a set of objects *or* a set of attributes with two maps:
  - ’ :: A↦B takes a set of objects to all the attributes which apply to all those objects.
  - ’ :: B↦A takes a set of attributes to all the objects which have all those attributes.

Concepts

- Iterating these two maps allow us to derive a concept from any old set of objects or attributes:

- The set A of objects determines a concept:

$$(A'', A')$$

- The set B of attributes determines a concept:

$$(B', B'')$$

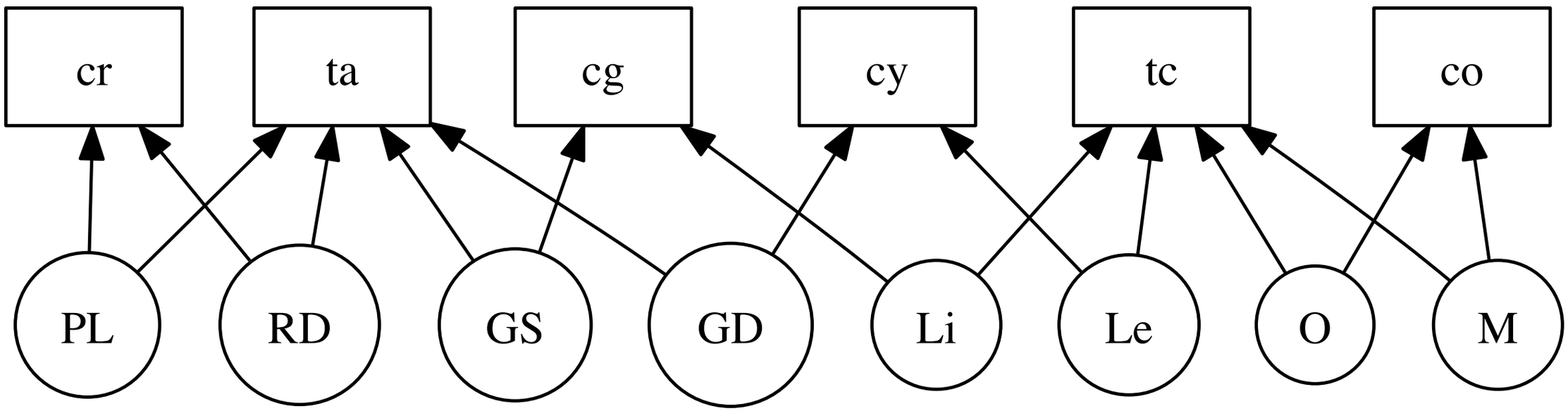
Example time!

Fruit

Name	Colour	Type
Pink Lady	Red	Apple
Granny Smith	Green	Apple
Golden Delicious	Yellow	Apple
Red Delicious	Red	Apple
Lemon	Yellow	Citrus
Orange	Orange	Citrus
Mandarin	Orange	Citrus
Lime	Green	Citrus

Fruit Context

Name	cr	cg	cy	co	ta	tc
PL	✓				✓	
GS		✓			✓	
GD			✓		✓	
RD	✓				✓	
Le			✓			✓
O				✓		✓
M				✓		✓
Li		✓				✓



Graph of I for the fruit context

Example 1

- X = {O}
- X’ = {co,tc}
- X’’ = {O,M}
- (X’’, X’) = ({O,M},{co,tc})

Name	cr	cg	cy	co	ta	tc
PL	✓				✓	
GS		✓			✓	
GD			✓		✓	
RD	✓				✓	
Le			✓			✓
O				✓		✓
M				✓		✓
Li		✓				✓

Example 2

- Y = {cr}
- Y’ = {PL,RD}
- Y’’ = {cr, ta}
- (Y’, Y’’) = ({PL,RD},{cr,ta})

Name	cr	cg	cy	co	ta	tc
PL	✓				✓	
GS		✓			✓	
GD			✓		✓	
RD	✓				✓	
Le			✓			✓
O				✓		✓
M				✓		✓
Li		✓				✓

Whither Lattices & Order?

- Lattice are structure which arises from a set of objects and an ordering on them. They are kinda sorta partially ordered sets which meet some additional criteria:

<S,≤>

- Example: any powerset $\mathcal{P}$(X) with the ⊆ relation forms a lattice.

- Another example: the set of concepts of any context form a lattice!

Concept Lattices

- A set of concepts form a lattice in two equivalent ways: based on extents or based on intents.

$$(A_1,B_1) \leq (A_2,B_2) \Leftrightarrow A_1 \subseteq A_2$$

$$(A_1,B_1) \leq (A_2,B_2) \Leftrightarrow B_1 \supseteq B_2$$

- This should hopefully make sense? A concept is “smaller” iff it has fewer (of the same) objects iff it has more (of the same) attributes.

Wither Functional Programming?

- I was starting to loose interest, even with chapter full of concepts I could almost get a handle on (excuse the pun) until I got to page 76.

  3.14 An algorithm for drawing concept lattices

- And it's a fairly simple algorithm too!

3.14 An algorithm for drawing concept lattices. Assume we have the cross-table of a context with the object set G down the side and the attribute set M across the top. The following instructions will generate a list of the extents of all concepts of the context in an order which is convenient for drawing the lattice of all concepts.

Step 1. Find all extents of the concepts of the context (G, M, I)**.**

(1.1) Draw up a table with two columns headed **Attributes** and **Extents**. Leave the first cell of the Attributes column empty and write G in the first cell of the Extents column.

(1.2) Find a maximal attribute-extent, say m'.

(1.2.1) If the set m' is not already in the Extents column, add the row $[m \mid m']$ to the table. Intersect the set m' with all previous extents in the Extents column. Add these intersections to the Extents column (unless they are already in the list) and leave the corresponding cells in the Attribute column empty.

(1.2.2) If the set m' is already in the Extents column, add the label m to the attribute cell of the row where m' previously occurred.

(1.3) Delete the column below m from the table.

(1.4) If the last column has been deleted, stop, otherwise return to (1.2).

Step 2. Draw the diagram with m **and** m' **labels.**

Start at the top of the diagram with one point labelled G. Work down the list of Extents in the table from Step 1. For each set S in the list, add an appropriately positioned new point to the diagram. Below the point corresponding to S list the elements in S. If S is an attribute extent

A. Initialise a table with one row [| G] to hold the concept-extents.

B. Loop: choose a maximal attribute-extent m’

1. If m’ is already in the table, add m to that row’s label.

2. Otherwise: add a new row [m | m’] and a new row for the intersection of m’ with each previous rows (don’t label these; skip any duplicates).

3. Delete m from the inputs.

C. Draw a diagram.

4. Each row is a node.

5. Label each node corresponding to an attribute-extent.

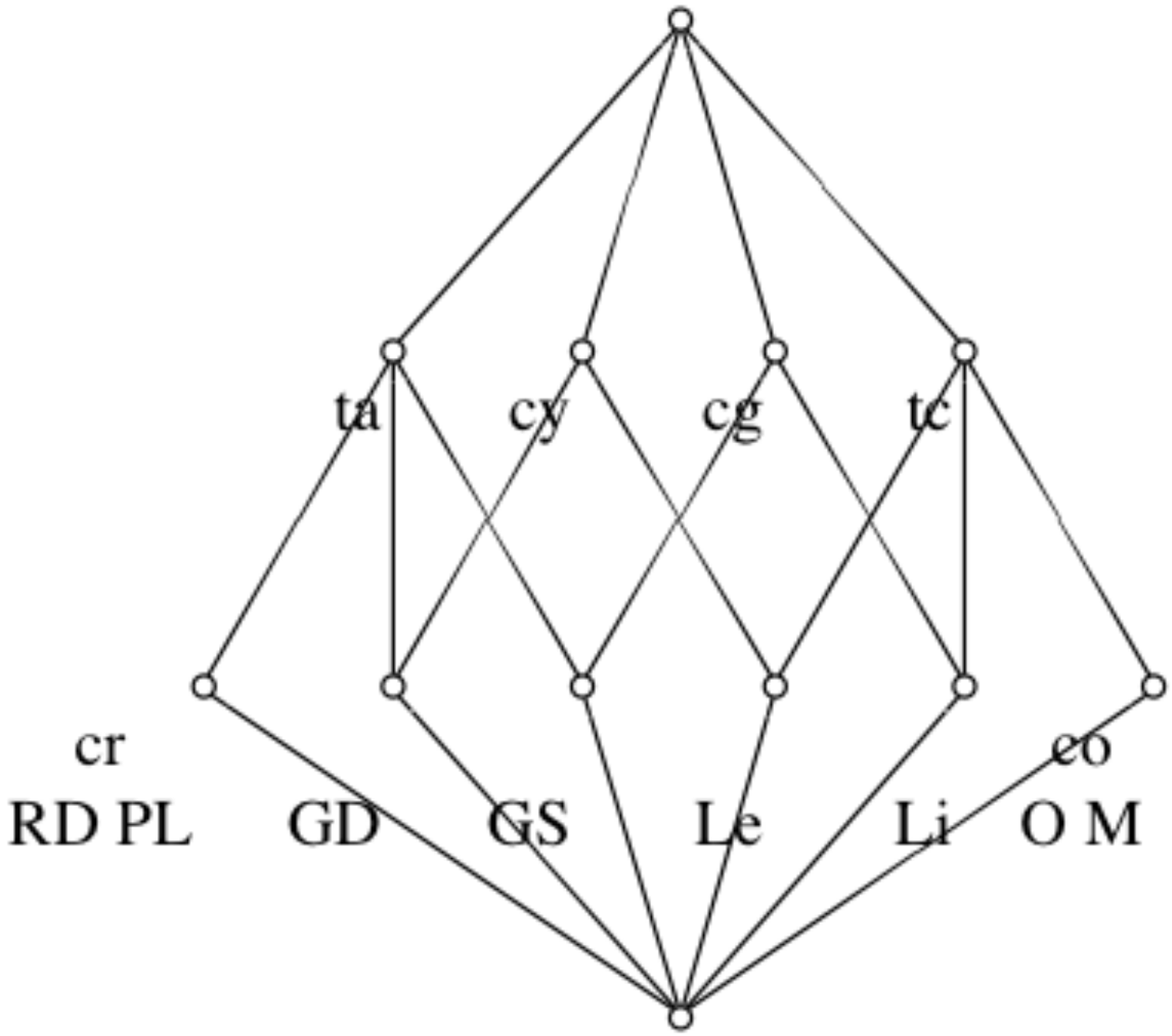
6. Label each node corresponding to the smallest extent containing each object.

Example

Name	cr	cg	cy	co	ta	tc
PL	✓				✓	
GS		✓			✓	
GD			✓		✓	
RD	✓				✓	
Le			✓			✓
O				✓		✓
M				✓		✓
Li		✓				✓

Attributes		Objects
1		GD,GS,RD,PL,Le,O,M,Li
2	cy	GD,Le
3	cg	GS,Li
4	ta	GD,GS,RD,PL
5		GD
6		GS
7	cr	RD,PL
8	tc	Le,O,M,Li
9		Le
10		Li
11	co	O,M
12		∅

Example



Building things in Haskell

FYI: This is where the “bad programmer” bit comes in.

Overview

- Using cassava to read input in CSV.
- Using containers and vectors to data structures.
- Using most brute-force-y and least efficient approach to every problem.
- Produces dot output which is rendered with Graphviz.

Straight-forward implementation

```haskell
parseContext :: Vector (Vector Name) -> (Context, Map ObjId Name, Map AttrId Name)
buildAETable :: Context -> AETable
generateGraph :: AETable           -- ^ Context lattice table.
              -> Map ObjId Name    -- ^ Map from object ID to name.
              -> Map AttrId Name   -- ^ Map from attribute ID to name.
              -> Text

main :: IO ()
main = do
  input <- BL.getContents
  case decode False input of
    Left err -> error err
    Right csv -> let (ctx, omap, amap) = parseContext csv
                     table = buildAETable ctx
                     graph = generateGraph table omap amap
                 in do
                   T.putStrLn graph
```

Core Algorithm

```haskell
-- | Construct the attribute/extent table of a context.
buildAETable :: Context -> AETable
buildAETable ctx = let g = V.foldl (\s v-> S.union s $ snd v) S.empty ctx
                       t = snd $ work ctx $ V.singleton (S.empty, g)
                   in t V.++ V.singleton (S.empty, S.empty)
  where
    work :: Context -> AETable -> (Context, AETable)
    work ctx table = maybe (ctx, table)
                       (\(a, ctx') -> work ctx' $ insertAttr a table)
                       (chooseMax ctx)

-- | Select the "maximal" attribute in the context.
chooseMax :: Context -> Maybe (Attribute, Context)
chooseMax ctx = fmap (flip vselect ctx) $ V.findIndex (f ctx) ctx
  where
    f ctx a = maybe True (const False) $
              V.findIndex (\b-> (snd a) `S.isProperSubsetOf` (snd b)) ctx

-- | Insert an attribute/extent into the table.
insertAttr :: Attribute -> AETable -> AETable
insertAttr (attr, ext) table =
  case V.findIndex (\r -> ext == snd r) table of
    -- Add the label to the existing row.
    Just j -> labelAttr j attr table
    -- Add a new row to the table.
    Nothing -> addIntersects attr ext $ extendTable attr ext table
```

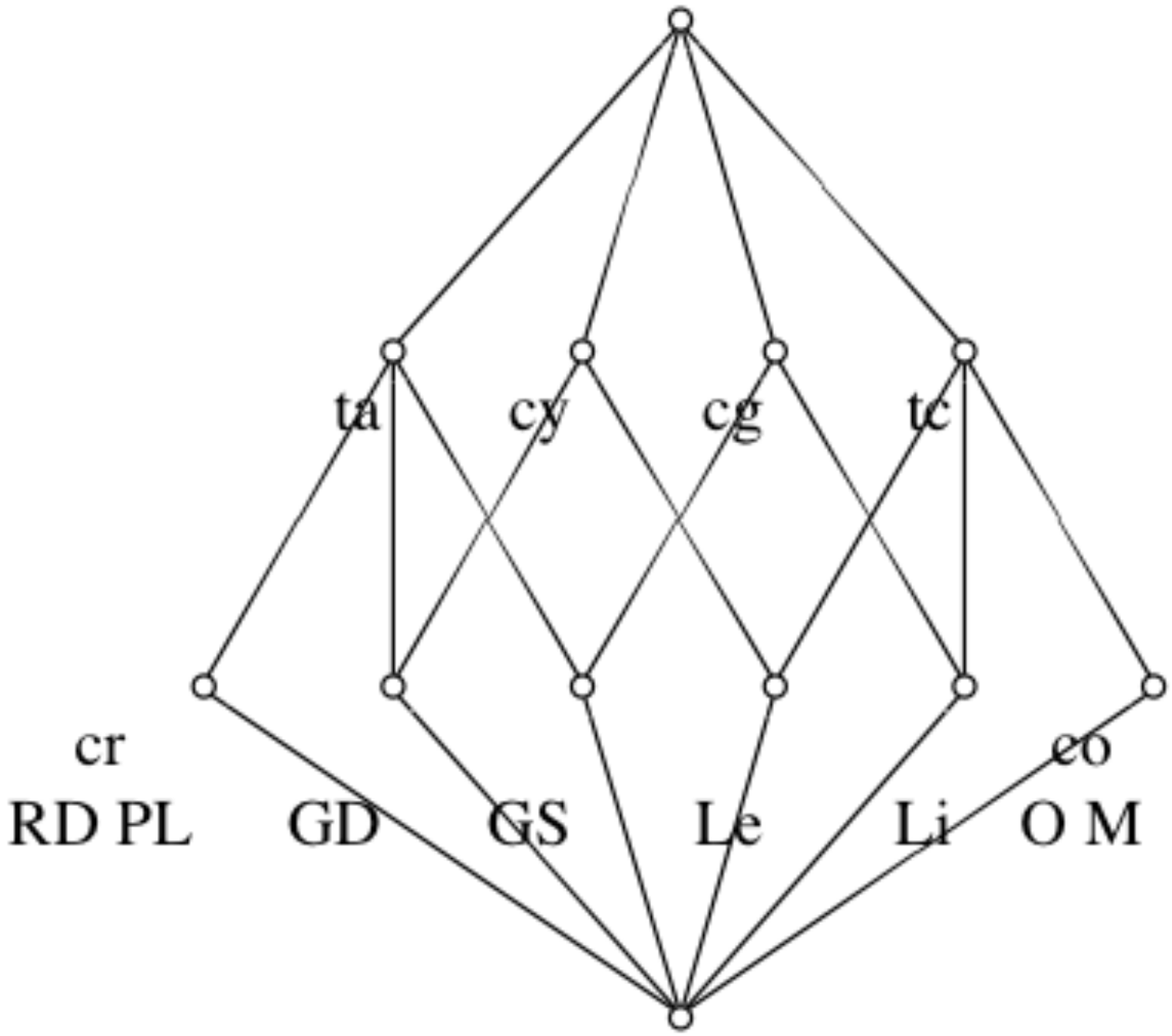
LOLWUT

```haskell
-- | Find the smallest set containing an element. NB: partial!
smallestWith :: Ord e => e -> Set (Set e) -> Set e
smallestWith e ss = let ss' = S.filter (S.member e) ss
                    in head $ sortBy myCmp $ S.toList ss'
  where myCmp x y = compare (S.size x) (S.size y)
```

```haskell
-- | Filter the elements @e@ of @s@ by whether or not @m@ maps @e@ to @s@.
filterSetByMap :: Ord i => Set i -> Map i (Set i) -> Set i
filterSetByMap s m = S.filter (\e -> (m M.! e) == s) s

-- | Find nodes which cover another.
covering :: Set Int                              -- ^ Extent to cover.
         -> [(Int, (Set Int, Set Int))] -- ^ Candidates.
         -> [(Int, (Set Int, Set Int))]
covering s ss = let ss' = filter (myCmp (1,(S.empty,s))) ss
                in filter (\c -> null $ filter ((flip myCmp c)) ss') ss'
  where myCmp = \(_,(_,s)) (_,(_,t)) -> s `S.isProperSubsetOf` t
```

Fruit Lattice



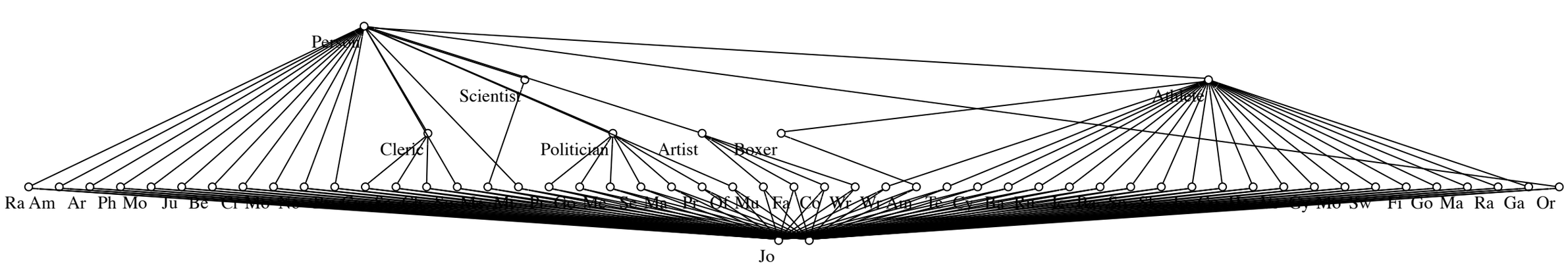
People in WikiDB

WikiDB is a set of DBs extracted from wikipedia metadata.
Extracted people and ~107 “types” applied to them.

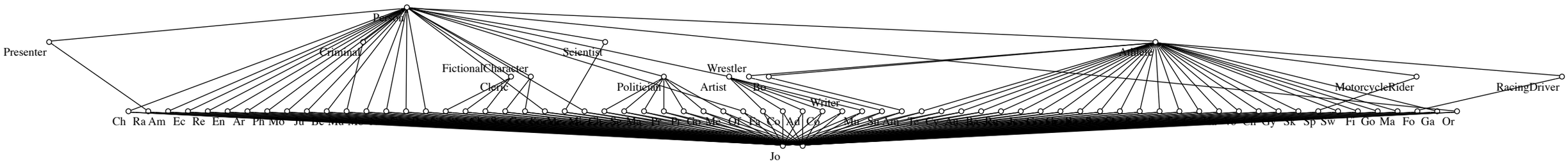
	CSV Input	DOT Output
1000	1000	348
5000	5000	474
“Complete"	72923	584

Data sets are the first *n* people which were convenient to extract from WikiDB data file.

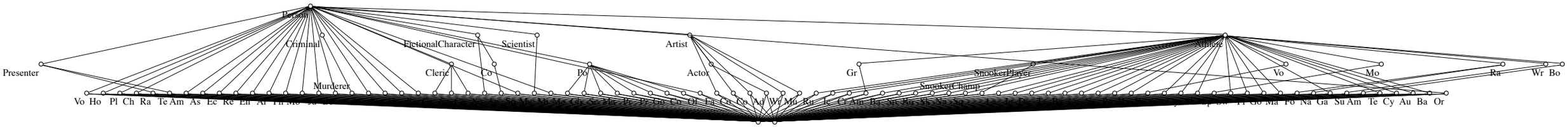
999 Persons from WikiDB



5000 Persons from WikiDB



72923 Persons from WikiDB



Improvements

1. Investigate better data structures. Set is probably not the best choice!

2. Read some RDF format or other instead of crazy CSV.

3. Space leaks!

4. Replace horrible brute-force code with smarter approaches.

5. Command line arguments to control output. Large graphs are utterly unreadable.

- Example of (4): calculate the graph for the whole lattice rather than the set of edges for each node.

References

- B.A. Davey, H.A. Priestly. Introduction to Lattices and Order (2nd). CUP.

- Wikipedia