Visualizing Polysyllogisms using Directed Graphs

December 12, 2025

1 Introduction

When Lewis Carroll was not writing *Alice in Wonderland*, he taught and wrote about logic, writing books such as *Symbolic Logic*. One of the logical topics he wrote on was the study of *soriteses*—a type of polysyllogism consisting of a sequence of chained implications that together yield a single final conclusion. Carroll produced many puzzles based on this structure, often designed to highlight surprising and often hidden consequences that arise from linking English premises.

To start, a syllogysm is a short implication chain, which put together provide a longer argument to be concluded from:

$$A \to B, B \to C \vdash A \to C$$

The natural deduction proof of this follows:

1.	$A \to B$	premise
2.	$B \to C$	premise
3.	A	assumption
4.	B	Modus Ponens (1)(3)
5.	C	Modus Ponens (2)(4)
6.	$A \to C$	$\to$-introduction (3-5)

A sorites extends this idea to any arbitrarily long chain of implications. When examining the Lewis Carroll sorites, the final conclusion is rarely obvious without formal on-paper reasoning as above. This motivates a computational approach to aid our solving and logical understanding.

The aim of this project is to take any of Carroll's sorites puzzles, formalize the natural-language premises into logical implications, and construct a structure for computing consequences. The completed program is published in `https://github.com/bklr/sorites-syllogism-graphed`

2 Carrol's Sorites Puzzle

The puzzle chosen for analysis is Problem 52 of Example 9 [1]:

Everything, not absolutely ugly, may be kept in a drawing-room; (1)
Nothing, that is encrusted with salt, is ever quite dry; (2)
Nothing should be kept in a drawing-room, unless it is free from damp; (3)
Bathing-machines are always kept near the sea; (4)
Nothing, that is made of mother-of-pearl, can be absolutely ugly; (5)
Whatever is kept near the sea gets encrusted with salt. (6)

Attempting to deduce the full chain's conclusion mentally is tedious and prone to error. Carroll's sorites often involved inherently illogical statements or a series of unrelated statements that further make the conclusion unintuitive. As a result, formalizing the natural-language phrasing into logical implications can make it easier to deduce.

3 Formalizing the Premises

Each English sentence is interpreted as a universally quantified implication of unary predicates. The six statements become:

$$\text{not ugly} \rightarrow \text{kept in a drawing-room} \quad (1)$$
$$\text{encrusted with salt} \rightarrow \text{not dry} \quad (2)$$
$$\text{kept in a drawing-room} \rightarrow \text{dry} \quad (3)$$
$$\text{bathing-machine} \rightarrow \text{kept near the sea} \quad (4)$$
$$\text{made of mother-of-pearl} \rightarrow \text{not ugly} \quad (5)$$
$$\text{kept near the sea} \rightarrow \text{encrusted with salt} \quad (6)$$

As proved earlier, the logical conclusion of the sorites can be derived by repeated use of Modus Ponens. However, carrying this out by hand quickly becomes lengthy, motivating a computational model.

4 Modeling the Sorites using Directed Graphs

To make the structure of the polysyllogism explicit and automate the process of deriving consequences, the English premises are encoded as directed implications inside a graph-based computational model. This program is written in Python and required designing an input language, implementing a parser, utilizing features of the NetworkX [2] library for graph construction and reachability analysis, and using PyVis [3] for interactive visualization.

4.1 Input and Parsing

Each puzzle is stored in a plain text file, one line per rule, using the format

```
A -> B.
```

For broader applications, the left-hand side may also contain multiple predicates joined by `&`, e.g:

```
A1 & A2 & ... & An -> B.
```

The parser implemented in the function `parse_implication_line` does the following operations:

1. Remove whitespace and ignore blank or commented lines.
2. Split the rule into left and right sides at the string `->`.
3. Split the left-hand side into individual predicates by `&` (if present).
4. Returning a list of the form `[A,B]` or `[(A1,A2,...,An),B]`.

This format allows any valid rule file to be processed, enabling multiple puzzles or broader logical systems to be graphed without modification to the program.

4.2 Constructing the Directed Graph

The implication structure is constructed by the `build_directed_graph` function, which converts the parsed implication rules into a `networkx.DiGraph` with the following correspondence:

- **Nodes**: predicates (e.g. `ugly`, `dry`, `bathing_machine`, `not dry`)
- **Edges**: directed implication relationships ($A \to B$)

If there exists a directed path from a node A to a node Z

$$A \to X_1 \to X_2 \to \cdots \to X_n \to Z,$$

then the implication $A \to Z$ is derivable by $n + 1$ repeated applications of Modus Ponens. Thus the consequence becomes computable by reachability and shortest-path algorithms. The result is a directed acyclic graph for most sorites structures (though cycles or contradictions can appear when inconsistent premises are included).

4.3 Computing Logical Consequences

Given a starting predicate, the program automatically computes all statements that follow from it. The function `analyze_node` returns two objects:

1. **Reachable predicates**: using `nx.descendants(G, start_node)`, this returns the set of all nodes N for which there exists a directed path from $start\_node \to N$.

2. **Shortest implication chains**: for each n in N, the shortest-path derivation is found using `nx.shortest_path(G, start_node, n)`.

Once a *start_node* is provided, the program prints (1) every reachable consequence and (2) shortest implication chain that leads each.

4.4 Automatic Sorites Conclusion

A defining feature of a sorites is the presence of a single longest chain connecting the initial premise to its final conclusion. To find this, the `find_longest_chain` function identifies, among all reachable nodes, the predicate node whose shortest-path distance from the starting node is maximal.

The corresponding path is the program's automatically generated "sorites conclusion":

$$A \to Z_{furthest}$$

where $Z_{furthest} \in N$ is the predicate node at the greatest graph distance from A. For the Carroll example (Problem 52 of Example 9 [1]), the program produces:

```
-------------------------
Longest implication chain in graph (Sorites Conclusion):
|   bathing-machine -> kept near the sea -> encrusted with ...
|   Concludes: bathing-machine -> not made of mother-of-pearl
-------------------------
```

This feature automates exactly the kind of reasoning that Carroll's puzzles intended readers to perform manually.

4.5 Visualization Using PyVis

To better understand the structure of the implications, the `networkX.DiGraph` is rendered using the PyVis visualization library. While NetworkX provides the underlying logical structure and reachability operations, PyVis provides an interactive, physics-based front end that makes the sorites chains easier to inspect. The visualization is constructed by the function `build_graph`, which performs the following steps:

1. Create an undirected copy of the directed graph for node transfer purposes, preserving the same set of nodes and edges. (Large NetworkX directed graphs can be more unstable in the physics engine).

2. Initialize a `pyvis.network.Network` object with a fixed canvas size, background and font color, and `directed=True` so that arrows appear on edges.

3. Enable a force-directed (Barnes-Hut) physics layout and interactive controls via `net.barnes_hut()` which calculates node positioning.

4. Add each predicate as a labeled node and each implication as a directed edge.

The resulting graph is written to an HTML file and opened in a browser, where the user can:

- drag nodes to separate clusters of related predicates,
- zoom and navigate to inspect particular regions of the graph,
- visually follow implication chains from any starting predicate to its consequences

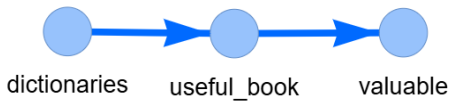

Figure 1: Simple implication visualization from Problem 2 of Example 7 [1]

5 Analysis and Contradiction

As shown in Figure 1, the visualization makes the fundamental structure of the polysyllogism apparent: one can immediately see how two end nodes of a chain are linked together. The visualization thus nicely complements the purely computational analysis of the sorites conclusions from Section 4.4 by providing an intuitive, interactive view of the polysyllogism.

5.1 Carroll's Contrapositive Subtleties

Although a sorites is traditionally defined as a chain of forward implications, many of Lewis Carroll's sorites puzzles—including Example 52—cannot be resolved by forward implications alone. In the example's sentences (2) and (3), there are statements made regarding dryness:

$$\text{encrusted with salt} \rightarrow \text{not dry} \tag{2}$$

$$\text{kept in a drawing-room} \rightarrow \text{dry} \tag{3}$$

Here, we see that certain premises are only able to interact with each other through their contrapositives. As a result, deriving Carroll's full sorites conclusion further requires drawing the implication chain through the logic implied within its sentences' contrapositives. The two resulting sorites chain and its contrapositive can be seen below in Figure 2.

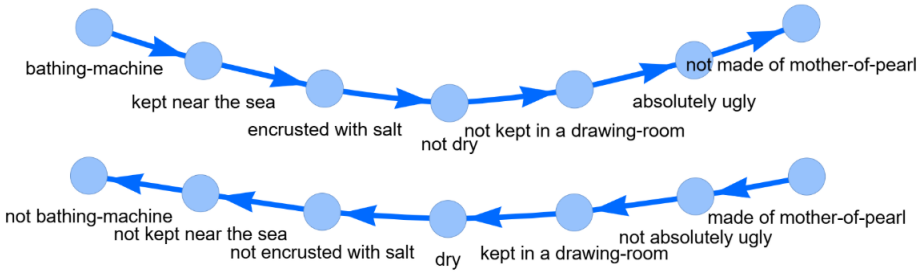

Figure 2: Problem 52's sorites and contrapositive implication chain

Though visual examination of the predicate `bathing-machine`, the graph reveals the same conclusion as Section 4.4:

$$\text{bathing-machine} \rightarrow \text{not made of mother-of-pearl.}$$

We are also now provided with its contrapositive,

$$\text{made of mother-of-pearl} \rightarrow \text{not bathing-machine.}$$

5.2 Identifying Structural Contradictions

One noteable feature of Carroll's soriteses is that they often contain an implicit contradiction that is never outright stated in the English text, yet becomes visible once both the implications and contrapositives are represented graphically. From the forward implications we obtain:

$$\text{bathing-machine} \rightarrow \text{kept near the sea} \rightarrow \text{encrusted with salt} \rightarrow \text{not dry}$$

while we also have:

$$\text{made of mother-of-pearl} \rightarrow \text{not ugly} \rightarrow \text{kept in a drawing-room} \rightarrow \text{dry}$$

The program's analysis already established the final sorites conclusion: bathing-machine $\rightarrow$ not made of mother-of-pearl but now the visual model makes clear why this must be true.

Suppose, for contradiction, that an object were both a `bathing-machine` and also `made of mother-of-pearl`. Then the two chains intersect in the middle and force the object to satisfy:

$$\text{dry} \wedge \text{not dry.}$$

This contradiction is not visible from the English sentences alone, but it becomes unmistakable once thee implication graph and its contrapositive structure are revealed—observe the center of the two chains in Figure 2. This contradiction can act as a proof of the sorites reasoning: because assuming a `bathing-machine made of mother-of-pearl` leads to an impossible thing, we can conclude its negation.

5.3 Another Example

To further illustrate that this program applies broadly beyond Problem 52, we include a second sorites puzzle of Carroll. Problem 60 of Example 9:

The only animals in this house are cats;	(1)
Every animal is suitable for a pet, that loves to gaze at the moon;	(2)
When I detest an animal, I avoid it;	(3)
No animals are carnivorous, unless they prowl at night;	(4)
No cats fails to kill mice;	(5)
No animals ever take to me, except what are in this house;	(6)
Kangaroos are not suitable for pets;	(7)
None but carnivora kill mice;	(8)
I detest animals that do not take to me;	(9)
Animals, that prowl at night, always love to gaze at the moon.	(10)

Formalized below with its contrapositives also entered into the program:

in-house → cat	(1)
loves-moon → suitable-pet	(2)
detested → avoided	(3)
carnivorous → prowl-at-night	(4)
cat → kill-mice	(5)
take-to-me → in-house	(6)
kangaroo → not suitable-pet	(7)
kill-mice → carnivorous	(8)
not take-to-me → detested	(9)
prowl-at-night → loves-moon	(10)

Automatically the Sorites Conclusion and generates Figure 3:

```
Longest implication chain in graph (Sorites Conclusion):
   |   kangaroo -> not suitable_pet -> not loves_moon -> ...
   |   Concludes: kangaroo -> avoided
```

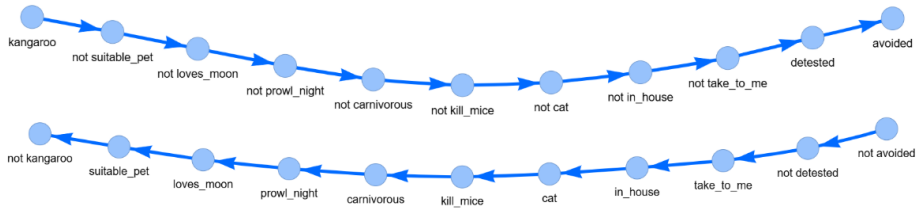

Figure 3: Problem 60's sorites and contrapositive implication chain

6 Extensions, Applications, and Limitations

6.1 General Applications

Although the computational model in this project was motivated by Lewis Carroll's sorites puzzles, the underlying method—encoding implication structures as directed graphs—can be extended far beyond. Using the directed graph representation to embed logic where: $A \to B, B \to C, \ldots$. Any domain in which statements, states, or entities depend on one another through implication relationships can be modeled using the same framework. Several examples include:

- **Course prerequisite systems.** The Boston College Computer Science curriculum requires courses be taken in specific orders, with each course often having a set of requirements. Graphical modeling can help students visualize their path and analysis can reveal paths they must take.

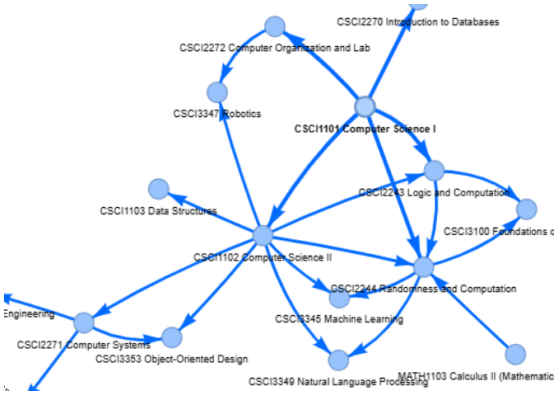

Figure 4: Simplified Boston College CS Course Graph

- **Computer security and access-control dependencies.** In the mock example below, the directed graph can reveal unintended access paths or contradictions in policy specifications—the same structural phenomena present in Carroll's puzzles.

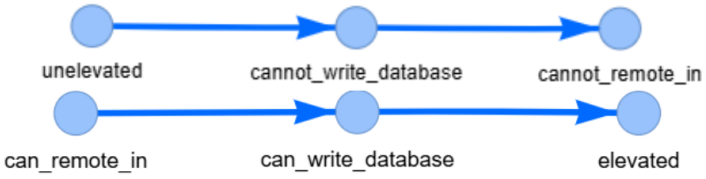

Figure 5: Mock Network Access Dependencies

- **Software package or module dependencies.** In package management systems, the installability of one module implies the availability of others. Directed graphs can expose dependency chains and impossible configurations

These examples show that the architecture created for analyzing a nineteenth–century logic puzzle applies equally well to modern computational reasoning tasks.

6.2 Logical Limitations

While implication graphs provide a powerful visualization and inference tool. Several limitations arise from this implemented process:

- **English-to-logic translation requires human interpretation.** Carroll's original sentences are phrased in plain English. The program cannot perform the translation to logical statements and each premise must be manually interpreted as a universally quantified implication between unary predicates. Different reasonable interpretations may yield different logical structures. (For our purposes, intended conclusions were verified in [1]).
- **No representation of quantifiers, disjunctions, or negations.** The system handles implications of the form $A \to B$, but cannot express premises such as "If A or B, then C," "Everything except X is Y," or nested quantified statements. Thus, the method is constrained to a particular segment of first-order logic.

6.3 Software Limitations

In addition to the logical limitations, the implementation has several technical constraints:

- **Contrapositives are not automatically generated.** While the system can easily incorporate contrapositives, the user must explicitly include them in the input file. Adding automated generation would remove human error and more accurately mirror formal logical deduction. Future work to handle negation can implement this feature.
- **The parser handles only conjunction on the left-hand side.** More complex structures (parentheses, nested statements, disjunctions, negations) are not supported.

Despite these limitations, the implication graph and analysis tool remains a flexible and extensible tool for analyzing chained logical structures. The general approach, utilizing directed graphs for visualization and automated logical deduction, makes it well suited for not just Carroll's logic puzzles but also a broad category of computational reasoning tasks!

7 Bibliography

[1] Lewis Carroll. *Symbolic Logic*, Book VII: Soriteses. 1896.
Public-domain digital edition accessed at:
`https://gutenberg.org/files/28696/28696-h/28696-h.htm#b7`

[2] NetworkX DiGraph Documentation.
Accessed at:
`https://networkx.org/documentation/stable/reference/classes/digraph`

[3] PyVis Documentation Tutorial.
Accessed at:
`https://pyvis.readthedocs.io/en/latest/tutorial.html`

- "Polysyllogism". Wikipedia.
Accessed at:
`https://en.wikipedia.org/wiki/Polysyllogism`

- George Wick. "Polysyllogisms & the Sorites". Amateur Logician.
Accessed at:
`https://amateurlogician.com/polysyllogisms-and-the-sorites/`