Alright spec time

The prose object model is a way of representing a legal document as a node on a graph (in the mathematical sense of the word). The document is defined by other nodes it references on a graph (this abstraction is described in Gabriel’s Monads, and is the abstraction that inspired PageRank as well). The cool thing about the prefix system (and the instantiation construct) is that it provides *a stronger differentiation of a node’s worldview.* Not only is a node’s view affected by its location in the graph (aka my perspective is shaped by location on earth. If I’m in Paris, I see the Eiffel Tower, if I’m in NYC, I see the Empire State), *but prefixing means that the node has a different etymology, and instantiation provides a different ontology*. Not only does the node have a unique view into the world – it has its *own unique world that only it inhabits.* A node’s world is defined by its location in the graph (its edges) *and* its prefixes. Let me continue the geographic analogy, except in this case I will add the concept of prefixing back in. Consider again two nodes – one Parisian, one New Yorker. Not only are their perspectives different spatially (one sees Paris, one sees New York), but they *literally live in different worlds*. The Parisian operates with a certain set of prefixes (French, Parisian, etc) that defines how it sees things. Consider the object my\_city. Both reference the file “city”, but one sees Parisian.city and the other sees New\_Yorker.city. Same object, different instantiation. They both reference the my\_city node, but instantiate it differently. **Same graph, same platonic objects, but different names, different instantiations.**

With that philosophical aside over with, I will now describe how I believe we should architect the program. I haven’t looked deeply at the Perl code (I’ve never used the language and don’t really have time to learn rn), but I’m guessing that it’s too tightly tied to the implementation of the Prose Object Model (specifically file based architecture). In my opinion, this makes the interface confusing – the user (even if this is only a library) should be allowed to *first think in Prose Object Graph*, and only then consider specific implementations.

Note: I am still not sure if I want to think about this functionally are object orientally yet, so forgive me for mixing paradigms / providing some ambiguity.

I propose that we begin by defining a set of function contracts that defines a Prose Object Model in the abstract. Each node shall be, in this document, hereafter referred to as ‘a Model’. Each Model contains a value (its text content which may be sprinkled with keys), and an edge list, and an edge lookup (where keys are linked to edges). Our first implementation should consider keys in the abstract (as a symbolic link to some value, not a string enclosed in braces), and values in the abstract (not as a string or filepath etc, but a pointer to *another Model)*. Let me note that it is also possible to conceptualize a node as *having no value unless it is a terminal node (one that contains a primitive, ie a string)*. Non terminal nodes will only contain a list of keys and a list of edges. In this case, the node that contains “my age is {jake.name}” will actually contain (in its edge list, in addition to other references) a reference to a node that contains “my age is “ and a symbolic link that contains {jake.name} (if this reference to this link cannot be found, it can be considered a reference to a string ‘{jake.name}’). Note that I have not yet thought out how we will represent the difference between a link determined by a key (which involves prefixing) vs a link w/ no key. It is possible that a keyed link can just be considered a subclass of Edge.

## Functions