Archives Package Management

Hans C. Suganda

21st June 2022

Contents

0.1 Rationale

As the Archives project gets larger and larger, it became necessary to organize the content. A good document must contain a high density of useful information, which is defined as the amount of useful information divided by page number. If the information density is too low, the reader will get disinterested.

Since the Archives project is meant to cater to a large variety of audience whose technical backgrounds vary from experts to beginners, the Archives project needs capabilities to display the "depth" of a certain topic. For example, a beginner might need an explanation to the most basic pre-requisite concepts to an advanced topic, but an expert might not need this since they already know from heart the basics of the topic.

The reader can be thought of something similar to a computational engine in logical capability, and the Archives can be thought of something akin to source code. What is needed now is a package manager that compiles the content/source code into useful readable format with all the necessary pre-requisites and is conflict-free.

0.2 Learning Model

How do humans really learn? From experience, it seems that academic knowledge should be treated like some kind of game. This seems odd. Really? A game? Let me explain.

A game is typically iterable over long periods of time. A game ideally should be non-deterministic because if it was deterministic then everyone would know the final outcome of the game based on the initial setup. If everyone could predict the final outcome of the game, there is no point in playing it and hence, the game is not iterable over time. This is an important quality that makes games "fun" because it is really difficult to tell who is going to "win" or "what" is going to happen. Academic knowledge is somewhat like this. Because of the complexities and difficulty within it, it is essentially impossible to tell what would happen in the future, which means that academic knowledge first the iterable criteria.

A game has to have rules. These rules are important because it constrains the players in a way such that their interactions become interesting. If there were no rules, then the game quickly devolves to a more predictable and less interesting state. Academic knowledge is constrained to be grounded in objective truth. The rules to academic knowledge is basically the rules of the universe itself, and the universe is unpredictable. This means investigators to these types of games have a high chance of discovering unexpected behaviours and patterns which makes academic knowledge very interesting. This means that academic knowledge satisfies the rules constrains.

A game typically involves multiple parties. A game that is played alone can be fun, but it would be more fun to play with friends. Academic knowledge is highly collaborative because of the sheer vastness of it. It is impossible for someone to know all the sum of human knowledge all at the same time, so having collaboration is the only viable way to proceed more effectively. This nature certainly encourages collaboration and hence ties the community aspect of games into academic knowledge.

0.2.1 Human Motivation

Humans are not machines. We have an intrinsic will and frankly learning a new subject is very difficult. Before we even begin to ask that a learner make the appropriate sacrifices in time and effort, we must show "why" they should. A vision that is powerful enough and inspiring enough is necessary to push through the difficulties of learning. It is not sufficient to just "command" that we all should learn a topic and sink so much time and rescources if there is not much to be ultimately gained.

0.2.2 Understanding New Concepts

From experience, it seems that there are 2 primary ways someone could learn a particular topic. The first one is to learn the "game" by being told explicitly what the rules are and then playing the "game". The second one is to learn the "game" by first playing it and based on what works and fails, infer the rules that dictate the "game". There are advantages and disadvantages to both methods.

Learning a concept by looking at the derivation is a very direct and general way to learn a particular topic. If this process is completed succesfully, the learner would have a complete picture and is able to understand all the possible cases. However, this method requires extreme attention to detail and requires alot of mental effort. Moreover, not everyone has the necessary pre-requisite to attempt such a direct assault on the topic.

Learning a concept by looking at the examples and then trying to "infer" the theory is a widely used method. This method typically requires less mental effort, but would fail in letting the learner know why and how the theory came to be. Moreover, if the example does not cover all possible cases, then it is possible that the learner later be exposed to a case not covered in the examples part and they would be left confused and unable to do anything.

It is also very common to learn the rules first by looking at the examples, and then reading the theory behind it. This is a good method provided examples do exist because it combines the initial speed and ease of learning by example, with the generality and completeness of the theory/derivation. A purely leraning by theory approach and purely learning by example approach seems inadequate.

0.2.3 Reinforcement & Adjustments

Human beings are creatures of habits and repetition. To really decrease the overhead of thinking about something and increase speed, humans need repetition at a particular task. This is very applicable to learning. Without applying what we learn, we would simply forget what was learnt and would have to rebuild from scratch again. This is why it is incredibly important to reinforce the concept using repetition. Understanding the concept is halfway, the other half is repetition to truly remember everything. Eventually with sufficient reptition, even the hardest concepts are as easy as flipping out like $2 + 3 = 5$. Remember that once upon a time a very long time ago, we all had difficulty with additions. There is no difference back then and now, only the names of the topics have changed.

No matter how perfectly we try to understand new novel information, it is very likely we would not understand it completely correctly at first. This is limited by alot of factors but primarily by language and context. A particular word like say "value" would mean many different things to many different people from a variety of life contexts. Therefore for a learner to truly grasp a particular concept as perfectly as possible, the last stage would be to correct all of the misconceptions about a particular topic.

The most effective way to figure out all the misconceptions is to try out practise problems and checking with the solution. Practising the problems force the learner to apply what they know so that any misconceptions can be revealed. Checking against the solution verifies any major discrepancies and mistakes. These mistakes can be used by the learner as feedback to correct their own understanding. By the end of the process, the learner should have a complete understanding of the concept with as few misconceptions as possible.

0.3 Structure of Archives Project

The Archives Project is essentially made of Nodes. Conceptually, a Node is the most indivisible unit of content. A Node should contain all of the following files, in the same directory. This makes sense because why would you want to spread a single Node's content over differing directories? Splitting the content this way allows us to focus on one aspect at a time. Apart from the metadata, all the files listed below should contain valid LaTeXcontent but without the pre-amble. This means the pure content is not compilable on its own but with the correct pre-amble, is compilable.

1. Metadata: This contains information on how this particular depends on other Nodes, and which other Nodes depend on this particular Node.
2. Rationale: Why should you learn this topic? Why is this topic important? What are the implications to this topic?
3. Theory file: The derivation, and the most abstract generalized theory the entire concept is based on. This is for learners who learn the game by being told what the rules are.
4. Examples: This is how the theory is applied. This is a realization of the Theory to a specific set of cases. This is for learners who learn the game by playing it first and then inferring the rules.
5. Problems: These are problem sets which are similar to the examples which helps learners train for a particular concept. These in conjunction to the solution can be used for extra examples.
6. Solutions: These are the exact solutions to the problems posed. This is important to give feedback on particular mistakes.

The file structure is translated directly from the human learning model. The rationale covers the human motivational aspect. Without the rationale, humans do not have anything they can hold on to in withstanding the difficulty of learning a new subject.

The Theory & Example are files which corresponds to the Understanding New Concepts part. Both files support learning the game by playing it or learning the game by reading its explicit rules. This makes the Archives a very versatile environment.

The Problems & Solutions correspond to the Reinforcement-Adjustment aspect. The Problems & Solutions ensure that learners can test their undertanding and be corrected in a constructive and helpful environment.

0.4 Algorithm of Pre-Processor

0.4.1 Data structure

The Node data is obtained from the Metadata file. Programmatically, the Node is an object which contains the information listed below:

- Topic Name: A unique name to identify this Node or content. This is the only way to identify a Node. No two Node can have the same name.
- Description: This is a short description of the topic contained in this Node. This
- Pre-Requisites: What are the other Nodes does this particular Node depend on? The other Nodes will be identified by their unique topic name.
- Builds-Into: What are other Nodes uses this particular Node as a pre-requisite? The other Nodes will be identified by their unique topic name.
- File List: What are the names of the files containing the Rationale, Theory, Examples, Problems, and Solutions? If this is left blank, then a default name is assumed.
- Organizational Hierarchy Level: Relative level of the Node. This can only hold the values between $1 - 7$ which corresponds to the default maximum organizational hierarchy LaTeXhas.

- Author: Who wrote this particular piece. This is not really necessary, it is just for fun. This can be left blank.

0.4.2 Algorithm Behaviour

In principle, the pre-processor is quite simple. Given the Node and certain user-specific settings, the algorithm will navigate through all of the possible directories and files all throughout the Archives to then fetch all of the Nodes which are pre-requisites to the Given Node. The algorithm will then dump all of those data into a fully fledge LaTeXfile at which the user can compile to see the final result. This way, if the user wants to make notes or whatever, they can directly modify the source code of the document they are seeing.

There are a few important realizations. Each run of the algorithm will choose Nodes such that their organizational Hierarchy Level is going to be consecutive. Not every single Node will have a complete File-List. This is because there will be some Nodes which are just sections, which will contain the rationale, but then not contain practise problems. There will be the subsubsections which are specific and may not contain rationale but then contains problems and solution so not every node will have one.

Why do we do it this way? because there are certain classes of nodes that rely on the same rationale or that its not possible for you to have questions from such a broad topic. For example you have differential equations, but which type do you want? Certainly the study of differential equations has a good rationale, but do you want practise problems literally placed at the section before the different types of odes are even explained?

The file hierarchy representation must resemble the section subsection subsubsection organization.

Suppose we have a run that produces Nodes with organizational hierarchy lists: 7, 6, 5, 4, the user should also have some way of reducing all of them by a set amount so we dont have parts and chapters for a small document. So the organizational hierarchy list after modification becomes 5, 4, 3, 2 or something like that.

You also need an automated way to form links and check links. So if topic A is pre-requisite to topic B, then topic B must be in the list of what topic A builds into.

We may also need a package checker, to make sure every tex file is referenced in the corresponding metadata and that every single file referenced in the metadata actually exists. This should spit out a few things: If a referenced file in the metadata doesn't exist, it will mention where the metadata is, where the file is predicted to exist, and throw a fatal error If a file is not referenced by any metadata file, then the file location would be given and a warning would be thrown.