B61E96AD21C0BD71	**Title**	**Conspectus 12022 Q1**
C21DCFD43C59CD93	**Subtitle**	**Chronical - I**
41FF07BFF9C7DD8A	**Author**	**Recursion Ninja**
1A0307F8DE0BA715	**Date**	**12022+084 Friday, March 25**
C80497A8815C5599	**Word Count**	**2712 (ERT 11 min)**
4C5C549C8B4E8433	**Code Lines**	**0**
22361BB6814E5FE5		
97A52FF56A1DCED4	**Formats**	**.adoc .epub .html .markdown .txt**

Quarter 1: Winter Solstice to Vernal Equinox

This is the first entry in my planned conspectus series, a sequence of quarterly reports summarizing my professional and academic activities. Each quarter is delineated between a solstice and an equinox. This first quarter of 12022 spans the Tuesday, December 21 winter solstice and the Saturday, March 20 vernal equinox.

Going forward, I plan to publish a chronicle entry shortly after each solstice and equinox. The purpose of the entries is to serve as a recountal for my own reference as well as a simple informational source for my family, friends, and peers to stay abreast of my professional efforts. I am pleased to start this series on a high note, with many notable events to share.

Doctoral Program Matriculation

First and foremost, I am compelled to report my most significant professional development. As I recently announced, I have accepted an offer to a Ph.D program starting in the fall of 2020. I will refer readers to the announcement for more details. However, I will add that this development will significantly shape the course of my professional trajectory for the next half decade and I am excited to discover where pursuing this new path will lead me.

Masters Thesis

Day to day, throughout the quarter I have been diligently working of completing my Masters thesis. The anticipated date of completion will be in mid April, with my defence at the end of April, and my final thesis submission to Hunter college occurring in May. My subsequent graduation is scheduled at the end of the summer 2022 academic term. I plan to elaborate on the details of my thesis in a subsequent blog post as it approaches it's completion. Until then, here is a truly brief

summary of my thesis.

The topic of my masters thesis explores the security of the TreeKEM protocol. My goal is to formally verify, using the exhaustive state-space model checking tools Promela and Spin, that the security properties of forward secrecy and post compromise security are inviolable via under the Continuous Group Key Agreement (CGKA) security game for the TreeKEM protocol. The methodology I employ is to model the CGKA game and the TreeKEM protocol in Promela and then use Spin to verify that, that no sequence of moves an attacker makes while playing the CGKA game can violate the specified security guarantees. It is quite an undertaking both in terms of mathematical precision, as semantic accuracy is paramount, and in software engineer, as the state-space explosion needs to be controlled. Though the progress thus far has been slower than desirable, I take solace in knowing that the scope of the task is greater than anticipated and that the result will effect nearly all secure group messaging apps.

AMNH Research Associate

For five years, from 2015 to 2020, I worked in the Wheeler Lab at the American Museum of Natural History (AMNH) developing a next generation phylogenetic graph analysis software platform. Unfortunately, the onset of the COVID-19 pandemic caused a lapse of funding which proved nearly fatal to the project, causing the development team to be dissolved before the software was completed. Since 2021, while not employed by AMNH, I have been an designated as a Research Associate while concurrently pursuing my masters degree.

This quarter, Wheeler lab has secured additional grant funding for the stalled project. We have been able to arrange for a temporary period of work over the summer of 2022 during which time the loose ends left from the abrupt lapse in funding can be wrapped up and leave the project in a state usable by other researchers. I am hopeful that the endeavors of this opportunity will be successful, allowing us to collaboratively publish and demonstrate the utility of this project, while simultaneously publicly recognising incredible amount of effort that the team in Wheeler Lab invested to bring the project to fruition.

Video File Shrinker

I have created and refined a Bash script for re-encoding video files in older codec formats to the most advanced available codec. Currently, the High Efficiency Video Coding (HEVC) is the most advanced and readily available video codec. There are more more advanced codecs recently published, notably AV1 and VP9, however there are not currently implementations of these formats which are efficient enough to be usable. The script recursively locates all video files in the specified

directories and then replaces all videos which are not in the HEVC format with an HEVC re-encoded version of the video.

The goal of the script is to save disk space, as the HEVC format takes noticeably less disk space to store the same video with the same quality that older codec formats. Experimental use of the script to re-encode videos archived on my external hard drives shows an average of roughly 50% less disk space usage. Results vary from file to file, with some having their size reduced by over 90% while others barely exceed 15%. Overall I am quite satisfied with the robustness and efficiency the script has exhibited thus far.

The project is:

1. Considered complete
2. Source code repository is publicly available
3. Released under the CC0 “Public Domain Dedication” license.

I-Blocklist Compiler

There is an additional Bash script I created for collecting and compiling multiple blocklists from the [I-Blocklist website]. The script queries the I-Blocklist API and compiles all *reasonable* blocklists which are exposed into a single, monolithic blocklist. The resulting blocklist is intended to be used by a [BitTorrent] client, preventing interaction with IP address ranges of known problematic actors. Results of the script appear successful in terms of compilation efficiency and resulting blocklist efficacy.

The project is:

1. Considered complete
2. Source code repository is publicly available
3. Released under the CC0 “Public Domain Dedication” license.

Haskell Code Quality Control

Haskell is by far my favorite programming language. There is little doubt that this stems from my near daily usage of Haskell for over eight years. I have not only used the Haskell language, but also contributed to the Haskell compiler GHC, the Haskell build tool Cabal, and numerous core libraries of the Haskell ecosystem. However, I have found myself unsatisfied with a few elements of my Haskell development experience. I have made some progress on identifying and remedying these deficiencies, but the process not near a usable state. An ideal outcome would be all of the troubles being addressed via git commit hooks and continuous integration checks. Below is a list of the pain points I find myself repeatedly irked

by and manually remedying.

1. Styling Cabal package description:

   Vexation: I want a standardized layout for package description files, I do not want to manually apply such a layout, and I am not satisfied with the cabal-fmt tool.

   Solution: Either invest in making cabal-fmt configurable or create a fork for my specific style.

2. Styling Cabal project:

   Vexation: Similarly, I want a standardized layout for cabal.project files, I do not want to manually apply the layout, and I have not found any existing styling tool.

   Solution: Create a configurable cabal.project file styler.

3. Styling Haskell source code:

   Vexation: I do not want to manually style my code nor do I like the existing (default) styles.

   Solution: Specify a style configuration for one of the existing configurable Haskell source code stylers.

4. Checking dead-code within Haskell source code:

   Vexation: The existing tool weeder only works when both weeder and the Haskell code has been compiled with *exactly* same version of GHC, requiring frequent and lengthy recompilation.

   Solution: Setup a continuous integration solution.

5. Checking documentation coverage of Haskell source code:

   Vexation: I would like to ensure that Haddock documentation exists for all exported functions and that module documentation exists for every module.

   Solution: Investing modifying Cabal or cabal-install to support documentation coverage checking.

6. Checking lower bounds of Cabal package description:

Vexation: I want to ensure that the project compiles with oldest compiler listed in tested-with field paired with the lowest bound listed for each dependency.

Solution: Improve my existing cabal-lower-bounds project.

7. Checking spelling of Haskell source code:

   Vexation: I would like for a spell-checker, rather than being directly on the source files, instead be applied to tokens on the abstract syntax tree while being aware of “camel cased” token names.

   Solution: Modifying an existing code styling tool to apply spell-checking transformations.