14740E8CBCE2C47B	**Title**	**Conspectus 12022 Q2**
6357561F92690F6C	**Subtitle**	**Chronical - II**
335E6D5D7E98D761	**Author**	**Recursion Ninja**
94163B1F07168661	**Date**	**12022+172 Tuesday, June 21**
C537DB144AD14939	**Word Count**	**1945 (ERT 8 min)**
29CE56E631883629	**Code Lines**	**0**
338AEDBB811ADECF		
C01304919B42AF0D	**Formats**	**.adoc .epub .html .markdown .txt**

Quarter 2: Vernal Equinox to Summer Solstice

This is the second entry in my planned conspectus series, recounting my quarterly professional and academic activities between a solstice and an equinox.

Masters Thesis

The academic journey of my masters program is soon coming to a close. A draft manuscript of my masters thesis has been submitted to my committee for review and my defense will take place in July. After a final draft is submitted to the School of Arts & Sciences, I will graduate from Hunter College in the City of New York and nearly immediately begin my doctoral work at The Graduate School and University Center of the City University of New York.

The results of my thesis are currently, and likely to remain, incomplete. Too little time remained between creating a correctly encoded model of TreeKEM and the deadline for submission to the college. Constraints of computing time required to verify the many combinations of model parameters exceeded the timeline for drafting, defending, and submitting the manuscript. Hence the results cannot yet be soundly applied to nearly all secure group messaging apps.

Despite this less than desirable culmination, I find that the experience overall has been greatly positive. The course of my masters program required I develop fluency in the language Promela as well as sophisticated competency with the Spin model checking tool. These skills are much less proficiencies of linguistics or engineering than they are a conceptual development in understanding the domain of verification. Consequently, I found the skills cultivated during my masters program to be surprisingly transferable between other formal methods tools which involve model checking.

Furthermore, the unfinished verification computations can continue after my thesis submission until all the model parameters have a completed result. At this time, the results can be re-tabulated and the work can be submitted for publication to a conference or journal venue.

AMNH Research Associate

Preliminary collaboration work has been resumed with my prior research team, Wheeler Lab at the American Museum of Natural History (AMNH). My efforts have been minimal during the conclusion of masters research, but what little time I could afford has been directed towards architecting an integration testing framework for the lab's next generation phylogenetic graph analysis software platform. The framework is written in Haskell, but designed to be extensible and maintainable by a team member without knowledge of the Haskell language. So far, the framework has been successfully deployed and the integration test-suite has grown to over 120 test cases!

To achieve success for our integration testing framework, I relied heavily on tasty; another testing framework which is both venerable and extensible. Individual integration tests are specified by indicating an execution script, a collection of input files, and one or more golden files. The specification components are then collected to create an integration test via the [tasty-golden] library which invokes the phylogenetic software platform under test to evaluate the script and generate the expected output files. Generated output is then compared to the expected output in the associated "golden" file, with any differences being reported as a test failure. The both the test specification interface and the technical construction of test suites are simple with Tasty.

The most crucial component to the integration testing framework's success is test case *inference.* To create a new test case, designers need only create a new directory containing all the specification components. When the integration test suite is invoked at runtime, it scans the test directory for all child directories which contain specification components. The integration test framework then constructs the Tasty TestTree dynamically at runtime and then execute all test cases in parallel. Furthermore, Tasty's withResource function enables some clever memoization of IO actions to reduce duplicate work across integration test cases as well as prevent race conditions during parallel test case execution. Overall the task has been an unqualified success.

Haskell Code Quality Control

In my prior conspectus, I describe some dissatisfactions I have with tooling surrounding the Haskell programming language. While I have not made addressing

any one of the listed grievance an active concern this quarter, happenstance has revealed some ameliorations. Here is a recountal of my original description along with the discovered (partial) remedy:

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.

Serendipitously, Marcin Szamotulski has been working on extending cabal with a new haddock-project command. I am hopeful that the culmination of their work will permit reporting documentation which is missing for any exposed definition in the generated Haddock documentation. Ideally, cabal project-haddock would support termination with an exit code other than EXIT_SUCCESS if one or more exposed definitions are missing a corresponding Haddock documentation annotation. Whether this desired termination behavior occurs by default or is enabled by a flag, it would greatly improve the new command's utility when utilized within the context of continuous integration.

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.

During the course of my masters thesis, I wanted to incorporate an automated spell-checking routine which was configurable in such a way that I could eliminate false positives from unrecognized domain specific terms, abbreviations, or acronyms not in a standard English dictionary. A cursory search of the GitHub Actions Marketplace did not disappoint. Ultimately I decided on selecting the @check-spelling/check-spelling GitHub Action. The Action is well documented, and open source. Most importantly, it supports adding "area dictionaries," dictionaries with additional words specific to a domain such as software terms, font names, or **Haskell**!