The 101haskell chrestomathy

http://softlang.uni-koblenz.de/101haskell/

Ralf Lämmel Thomas Schmorleiz Andrei Varanovich

University of Koblenz-Landau, Software Languages Team, Germany

Extended abstract

In philology and linguistics, the term *chrestomathy* refers to a collection of sample texts in one language designed to be useful for learning the language by demonstrating some language aspects such as language development or literary style.

In programming, the term *program chrestomathy* refers to a collection of sample programs in one or more programming languages designed to be useful for learning programming (or becoming more proficient in programming) by demonstrating some programming language aspects such as comparison of programming style, expressiveness, and applicable programming techniques in one language or across different languages.

More broadly, the term *software chrestomathy* [1] refers to a collection of software systems relying on one or more software languages as well as any number of software technologies; a software chrestomathy demonstrates aspects of programming and software development. When compared to a program chrestomathy, a software chrestomathy collects systems rather than programs, thereby possibly covering additional details such as building, testing, and sample data.

The 101haskell chrestomathy is a collection of tiny or small Haskell-based software systems designed to be useful for learning functional programming in Haskell. The collected systems present Haskell-based solutions to a number of general system requirements as defined by the 101companies project [1]. Also, the systems exercise alternative applicable programming techniques and technologies (e.g., libraries) for the requirements. The 101haskell chrestomathy is the Haskell-specific sub-chrestomathy of the 101companies chrestomathy which covers dozens of programming and software languages. Following the terminology of 101companies, the collected systems are called *contributions*, thereby emphasizing the community aspect of collection.

It happens that Haskell has played a special role in the 101companies project, i.e., Haskell has been used to bootstrap and demonstrate various capabilities of 101. (To a slightly reduced extent, this is also true for Java.) Here is a partial list of such capabilities:

Code management The tiny or small systems ('contributions') are all readily available from a GitHub repository, with Cabal-based dependency tracking and building and testing, thereby supporting software engineering best practices and collaborative development and maintenance.

Content management The documentation of each contribution is readily maintained on the 101wiki which also aggregates knowledge about software concepts, languages, and technologies. All content is managed in several namespaces with a metamodel designated to each namespace.

Semantic properties With many software concepts (e.g., 'monad'), software languages (e.g., 'XML' or 'SQL'), and technologies (e.g., 'GHC' or 'SYB') involved, it is important to maintain semantically strong links between all these entities. Thus, links are qualified by a property (predicate). For instance, a specific link may express that a given system *uses* a certain technology.

Wikipedia integration In the 101companies project, we do not try to define all the encountered software concepts anew; instead, the objective is to connect to existing knowledge resources. In particular, we apply a scheme of curation such that concepts are linked to Wikipedia pages while qualifying the level of curation: 'same as', 'similar to', 'not the same as'.

Textbook integration With some textbooks being accessible online, it becomes possible to link wiki content to textbook content so that users may take advantage of integrated knowledge. We use a specific framework for text mining and presenting such links; two Haskell textbooks are readily linked.

Fragment location Documentation may also refer to code at the fragment level (e.g., individual function or type declarations). In this manner, we separate classic program documentation (which is part of the code) from the higher-level discussion of systems on the wiki. There is the language-parametric notion of fragment location which supports references to code units and their retrieval or rendering.

Themes (sub-collections) All the Haskell-based systems are organized in a number of themes addressing different interests (stakeholders), e.g., generic functional programming. Haskell-based systems also participate in themes that are not Haskell-specific, thereby demonstrating the 'Haskell way' of addressing some problem, e.g., parsing or web programming.

Code similarity management It is important to manage (to understand, to maintain) the code similarity across systems using the same language. For instance, several contributions may use the same or a similar data model and such reuse should be discoverable and maintainable (in the view of evolution). We investigate special forms of clone detection and version control to this end.

Didactic structure The different systems need to be explorable in a systematic manner to be more useful for learning. The 101haskell systems are organized along several orthogonal dimensions, e.g., in a sequence of lecture scripts. Further, the individual systems refer to each other to explain similarities and differences.

Acknowledgment

We are grateful to several people (fellow '101ers') who helped on the technical part of this work—specifically Kevin Klein, Olexiy Lashyn, Martin Leinberger, Sebastian Jackel, and Arkadi Schmidt.

References

[1] J.-M. Favre, R. Lämmel, T. Schmorleiz, and A. Varanovich. *101companies*: a community project on software technologies and software languages. In *Proc. of TOOLS 2012*, volume 7304 of *LNCS*, pages 59–74. Springer, 2012.