

Chapter 1

Evaluation and Discussion

This chapter analyzes and discusses the research results.

Section 1.1 presents the main findings that are connected with the research purpose. Section 1.2 interprets how the research results support these findings. Section 1.3 contrasts my findings with the results of the past researches. Section 1.4 explains the discrepancies between my results and findings. Section 1.5 describes the limitations of my research. Section 1.6 lists the unexpected findings. Section 1.7 suggests the possible applications of my research findings.

I Key findings

The main finding is that it is possible to build a Haskell eDSL as a library that allows for declarative spreadsheet construction. A user specifies the connections between the spreadsheet elements, and the library functions produce the elements layout automatically. Despite being simple, the language requires the knowledge of Haskell at least at the beginner level. That is why, the target audience of the language is Haskell programmers. They are expected to generate the spreadsheets and pass these spreadsheets to non-programmers.

Another finding is that such an eDSL provides the means to construct statically typed formulas. This feature allows for typechecking the formula expressions at compile time. Such checks prevent runtime errors, such as invalid formula arguments, in spreadsheet applications.

The third finding is that it is possible to use the existing Haskell features in the programs written in the new eDSL. First, formulas can be curried, composed, and applied to ranges of values and cell references. Second, algebraic data types can be used to model the value domains. Third, the `do-notation` works because the eDSL has a monadic interface.

II Results and findings

First, the result of my research is the `clerk` Haskell package. It is publicly available on GitHub [1] and Hackage [2]. Haskell programmers may discover the package there, read through the examples from the package description and use `clerk` in their programs. The examples are based on simple problems and demonstrate most features of the eDSL.

Second, the type checking of formulas works at compile time. A user first specifies the formula signature - a name and the types of arguments. Later, the user can apply a formula to arguments.

Third, the examples from the package description demonstrate how to apply certain Haskell features to write type-safe monadic code using `clerk`.

III Previous research

Currently, the major spreadsheet applications provide limited tools for declarative type-safe construction of spreadsheets.

A. User-defined data types

Firstly, Microsoft Excel provides creation of user-defined data types [3]. A user may group the columns of values into records that represent user-defined types. Additionally, they may use a user-defined data type as a type of a field of another user-defined data type.

In contrast, `clerk` allows to type-safely group values into Haskell records. This approach allows for building type-safe composite records before importing them into a spreadsheet application.

B. User-defined functions

Microsoft Excel introduced the `LAMBDA` function [4] that allows for user-defined functions. Google Sheets also provides the `LAMBDA` function [5] that should immediately be applied to a value. Thus, in this section, I focus on the `LAMBDA` function from Microsoft Excel.

First, the `LAMBDA` function can be recorded and shared between spreadsheets. Second, this function allows for recursion and usage of other functions, including user-defined ones. Third, user-defined functions are dynamically-typed. The argument types can be specified as comments to a recorded user-defined function.

In comparison to `LAMBDA`, `clerk` allows a user to declare functions in Haskell. First, these functions can be imported into other Haskell modules or programs. Second, these functions may represent compositions of functions from the target spreadsheet system, including user-defined ones. Third, these functions are statically typed. Their arguments may be documented using Haddock.

C. Declarative layout

Currently, Microsoft Excel has support for automatic layout of data upon importing it [3]. However, layout customization still should be done manually.

The `poi` library for Python can write spreadsheets based on user data [6]. The library seems to not allow to specify the connections between new elements and previously built elements.

Such functionality, on the other hand, is present in the Kotlin library `Geschikt` [7]. The library provides a declarative API to building Google Sheets. A user may create an cell object and then access its reference in the expressions that follow. The downside is that there seems to be no way to use a reference outside of a parent element.

In `clerk`, it is possible to get a reference to a specific cell. It is the user's responsibility to return that reference from the function where the cell is created. The cell references can be

IV Discrepancies

V Limitations

First of all, currently, the eDSL is not usable by non-programmers. Perhaps the implementation of a graphical user interface to the eDSL may solve this problem.

Next, another problem is that the library generates spreadsheets using the limited number of formatting tools provided by the underlying `xlsx` library. Thus, if a user edits a generated part of a spreadsheet, these edits will be removed upon the next spreadsheet generation. This limitation can be overcome by copying just

the data from the generated parts into new sheets and applying formatting there.

VI Unexpected findings

VII Outer applications

The `clerk` library allows to produce correct by construction spreadsheets. I suppose that this property can make the library useful in real world systems.

First, this library can be used as a part of data pipelines. The program using the `clerk` library can generate data-transforming spreadsheets. There will be regions for input data, output data, and formulas. The formulas will act onto input data to produce output data. Other programs may write to or read the outputs from such data-transforming spreadsheets.

Second, such a library may be useful in areas that require correct computations. As far as I know from Haskell community chats, some financial and biotechnological organizations rely on Haskell. Moreover, some people use Excel for recording the results of scientific experiments.

Third, it is possible to use `clerk` for generating reports. The library provides basic styling capabilities. When the library is used with a Haskell repl, the resulting spreadsheets can be generated rapidly. Thus, the spreadsheet editor may quickly observe the changes.

Fourth, the library may be used by pixel art fans. It is quite trivial to fill the table cells with colors of pixels of an image.

Fifth, the library can be used for teaching type-level programming in Haskell. More specifically, the library includes modules that provide type-level parsing capabilities. These modules enable compile-time cell address and color values

checks.

...

Bibliography cited

- [1] D. Danko, *Clerk*, original-date: 2022-12-26T16:42:25Z, Mar. 29, 2023. [Online]. Available: <https://github.com/deemp/clerk> (visited on 05/19/2023).
- [2] “Clerk,” Hackage. (), [Online]. Available: <https://hackage.haskell.org/package/clerk> (visited on 05/19/2023).
- [3] “Create a data type (power query) - microsoft support.” (), [Online]. Available: <https://support.microsoft.com/en-us/office/create-a-data-type-power-query-a465a3b7-3d37-4eb1-a59c-bd3163315308> (visited on 05/19/2023).
- [4] “LAMBDA function - microsoft support.” (), [Online]. Available: <https://support.microsoft.com/en-us/office/lambda-function-bd212d27-1cd1-4321-a34a-ccb254b8b67> (visited on 05/19/2023).
- [5] “LAMBDA function - google docs editors help.” (), [Online]. Available: <https://support.google.com/docs/answer/12508718?hl=en> (visited on 05/19/2023).

-
- [6] R. Wang, *Poi: Write excel XLSX declaratively*. Version 1.0.1. [Online]. Available: <https://github.com/baoshishu/poi> (visited on 05/19/2023).
- [7] “Sikrinick/geshikt at androidexample365.com,” GitHub. (), [Online]. Available: <https://github.com/sikrinick/geshikt> (visited on 05/19/2023).