**Frege IDE**

Master Thesis

*(Please note, this is a work in progress, the text and the graphical layout will be updated)*

**Outline**

* Chapter 1: Introduction
  + Motivation / Introduction *(what was already done, why this work even exists, how this relates to other IDEs, what is Frege)*
  + Goals *(What I strived to achieve in this work, what subset of features should make / made it here, why exactly this, how this relates to projectional IDEs – MPS, why choose a projectional IDE, what is MPS – very shortly)*
  + Outline *(what other chapters are here, what can be found there)*
* Chapter 2: Meta-programming system / JetBrains MPS
  + - *what it is, what it can do, links & references*
    - *difference between projectional and classic text-based IDEs, how this relates to MPS, what makes MPS different*
    - *examples (images), what it would be like to use MPS, so the reader has a clearer picture*
    - *working with MPS, structure, editor, constraints, behaviour, checking rules, typesystem, textgen – generic explanation + exemplar language (not necessarily tutorial, but the explanation should be clear and how it would be used)*
      * *Probably shouldn't go into too much detail for typesystem and editor? - will be explained again on Frege-IDE (how it was done + options and possibilities)*
* Chapter 3: Frege
  + *what is Frege*
  + *Frege vs. Haskell*
  + *Frege language, examples, and what we decided to support in 'Frege-IDE'*
* Chapter 4: The grammar
  + *Official Frgege grammar, or references (appendix?)*
    - *This is too abstract. Maybe pick only some interesting parts?e.g. “this is usually solved as...”, “on this level it means this and this is supported”...?*
  + *Subset related to what we decided to support in “Frege-IDE”*
    - *(Analysis, what parts of the grammar we decided to cut off, what couldn't be cut off, hot it relates to everything)*
* Chapter 5: Frege in MPS
  + *Grammar transformation for MPS structure aspect, design of the Frege structure, how and why (analysis + design decisions)*
  + *Editor*
    - *“Visuals” (FunctionName FunctionParameters = Expression)*
    - *Editor transformation and substitution actions*
    - *Lacking documentation on the MPS part.*
  + *References, scopes*
    - *(Constraints + behaviour)*
  + *Typesystem*
    - *Expression evaluation*
    - *TBA*
* *Chapter 6: Evaluation*
  + *What we strived to achieve, again? Recapitulation?*
  + *User-friendliness of the “Frege-IDE”, against classical text-based IDEs, advantages and disadvantages*
    - *(Should be probably mentioned user's time investment into learning to use the IDE)*
  + *Limitations*
    - *Something couldn't be done easily, e.g. Enter key-press does not always create a new line, ambiguity*
    - *Not all transformations are possible, e.g. rewriting f x y = x + y to f , x, y :: Int -> Int -> Int (by removing = and adding , , ::) is basically impossible to cover – it is not a text editor*
    - *Intentions – for the example above intention is an option to cover such transformation*
    - *(Mention built-in JAVA-like language, that also has these limitations)*
  + *Experience*
    - *What I found MPS lacking about, flexibility, not much detailed documentation*
* *Conclusion*
  + *Are projectional IDEs good for functional languages?*
  + *Is “Frege-IDE” usable?Future work, possible extensions.*
* *References*
* *Appendix*
  + *Frege formal grammar*
  + *Source code description*
  + *Examples*
  + *User manual for “Frege-IDE”*
* Chapter 1: Introduction
* Motivation / Introduction *(what was already done, why this work even exists, how this relates to other IDEs, what is Frege)*

(Classic intro,a lot of wiki references to say the obvious and well known.)

(what are IDES) (<https://en.wikipedia.org/wiki/Integrated_development_environment>) Integrated development environments (abbreviated IDEs) are a set of software applications that provide tools and facilities to computer programmers. They greatly ease the process of software development, providing features like intelligent code completion, syntax highlighting, build automation tools, debugger, etc. (…add anything??)

(what is Frege) (<https://en.wikipedia.org/wiki/Frege_(programming_language)> Frege, named after the German mathematician, ~~logician and philosopher~~ [Gottlob Frege](https://en.wikipedia.org/wiki/Gottlob_Frege), is a functional language heavily based on Haskell (add ref. link) for the Java virtual machine. It is considered a Haskell dialect, sometimes called “a Haskell for the JVM”. (…todo)

known IDEs

There are several IDEs for Haskell, not many specifically built to support Frege. Examples can be found at <https://wiki.haskell.org/IDEs>, most of them providing mainly syntax highlighting, macros and project management features, while some also provide more advanced features like code completion or type evaluation and inspection.

That being said, all of the notable IDEs are text-based, (add explanation?

See <https://en.wikipedia.org/wiki/Structure_editor>

…most source code editors are instead text editors with additional features such as syntax highlighting and code folding, rather than structure editors. The editors in some integrated development environments parse the source code and generate a parse tree, allowing the same analysis as by a structure editor, but the actual editing of the source code is generally done as raw text.

). In this work, however, we try to approach the topic from a different view and try to design a projectional editor (/IDE) specifically for the Frege programming language.

(what is projectional editor) quoting <https://en.wikipedia.org/wiki/Structure_editor>

A projectional editor (aka structure, structured editor), is a document editor that is cognizant of the document’s underlying structure. They are usually used to edit hierarchical or marked up text, computer programs, diagrams, and any other type of content with clear and well-defined structure. While for the most computer programs a conventional text-based IDE may be more suitable, for specific programming languages, especially DSL (<https://en.wikipedia.org/wiki/Domain-specific_language>) a projectional editor might prove to be a more effective tool.

<https://martinfowler.com/bliki/ProjectionalEditing.html> *- additional info sources*

In this work we want to design a projectional editor supporting a certain subset of the (features of) Frege programming language and examine what are the advantages and disadvantages of such approach over creating a conventional text-based IDE.

* Goals (What I strived to achieve in this work, what subset of features should make / made it here, why exactly this, how this relates to projectional IDEs – MPS, why choose a projectional IDE, what is MPS – very shortly)

As was already mentioned in this paper, in this work we design a projectional editor for a subset of (features of) Frege programming language.

We have chosen JetBrains® *(legality, trademarks, etc.? check)* MPS platform as an underlying tool for designing our IDE. (briefly - why:) MPS (standing for “Meta-programming system”) is an open source software solution allowing developers and language designers to create different kinds of projectional IDEs. It is a so called language workbench (A language workbench is a software development tool designed to define, reuse and compose domain-specific languages together with their integrated development environment. - <https://en.wikipedia.org/wiki/Language_workbench>)

It allows for creating both simple and complex languages, especially DSLs, while also allowing extending existing ones, when the languages available do not meet the needs of a developer. *(quoting MPSLW I.)* MPS has a large set of features allowing for designing editors which closely resemble those from conventional, text-based, IDEs.

**Frege features**

Frege, based on Haskell language, has rather many syntactic (and semantic) constructs for this work to be able to include them all. We, therefore, focus only on the parts of the Frege that make it so popular functional language. ‘Syntactic sugars’ are for the most part omitted, as well as monads, which make Frege appear less functional and a more of an imperative programming language. To include more advanced features, like context help (references) and type system checking, we also had to keep the complexity of the work reasonably small (feasible) and thus concepts like classes and instances had to be omitted as well.

The following part includes the high level concepts of the supported Frege features.