Improving the Usability of Mondriλn, a Visual Esoteric Functional Programming Language

Elvedi, Sven and Strolz, Timo

Fall term 2025

Project Documentation

Advisor: Prof. Dr. Farhad D. Mehta

Department of Computer Science
Eastern Switzerland University of Applied Sciences (OST)
Campus Rapperswil

Abstract

The aim of this project is to make the lambda calculus more accessible through interactive visualization. A previous project developed Mondriλn as a proof of concept CLI application that generates Mondriλn images from lambda terms and vice versa. This project builds on the initial proof of concept by re-developing it as an interactive web application (Mondriλn+). Improved aspects are the visual representation of nested abstractions and parenthesized terms. Additional features include customizable visualizations, an α-conversion equivalent transformation for the images, an interactive web interface and an enhanced language specification for the images.

Mondriλn+ is written in Haskell and uses Nix in combination with Cabal as its build tool. The production code gets compiled to WebAssembly. This enables the development of a type safe and reactive UI with reproducible builds and high portability. The Mondriλn language specification was redesigned to enable greater customization, stronger invariants, and an improved user experience. Among the evaluated Haskell web frameworks, Miso was chosen for its simplicity and existing documentation. The core concept of the application remains the pure lambda calculus with an extended intermediate representation containing visual information on how to render the resulting images.

Mondriλn+ can generate images from textual lambda calculus terms and vice versa. The application can perform β-reduction on Mondriλn+ images and on their corresponding textual representations. It supports α-conversion equivalent transformations on images, implemented as color changes. Users can either customize Mondriλn+ images implicitly by the way they enter textual terms and explicitly by resizing individual elements. The application can be used to teach and learn the lambda calculus in an interactive and visually appealing way. The current version of Mondriλn+ has potential for further development. Ideas for additional features include for example animations of β-reduction or scanning printed versions of Mondriλn+ images.

Management Summary

Background

The lambda calculus is taught in functional programming courses, but students often find the abstract mathematical notation challenging to understand. A previous project created Mondriλn, a visual esoteric programming language, and a CLI tool for it that converts mathematical lambda expressions into Mondriλn images and back into lambda-terms. However, this tool was command-line based and lacked a user interface therefore. This project extends Mondriλn into a user-friendly web application that brings lambda calculus to life through interactive visualization. The application transforms complex mathematical expressions into intuitive images, making the learning process more engaging and accessible to students. For simplicity, this project will refer to the initial proof of concept version as Mondriλn and to the new version as Mondriλn+.

Approach

Mondriλn+ is a web-based application that requires no installation and runs within the web browser. The application was built using Haskell, a programming language known for its reliability and ability to catch errors before the code even runs, ensuring the application is robust and stable. A major focus was improving how lambda expressions are visually represented. The visual language was redesigned to better support complex nested functions and parenthesized expressions, making them easier to understand at a glance. The user interface was designed to feel natural and responsive, users can interact with diagrams by either clicking or dragging. The application combines text and visual editing seamlessly. Students can type mathematical expressions and see them instantly rendered as diagrams. Interactive color coding automatically highlights all variables of an expression that are related, helping students trace through complex relationships without getting lost.

Results

Mondriλn+ successfully delivers an easy-to-use platform for exploring the lambda calculus:

- Users can input mathematical expressions and create visual diagrams of them
- The application visualizes step-by-step computation, helping students understand how expressions are evaluated
- Interactive color coding highlights related parts of expressions, making complex relationships clearer
- Users can resize the visual elements to find the representation that works best for them

In practice, Mondriλn+ makes lambda calculus tangible. Instead of struggling with abstract notation on paper, students can immediately experiment, see results, and build intuition through interaction. The visual and interactive nature of the application transforms a purely mathematical concept into something students can explore, manipulate, and understand through hands-on learning.

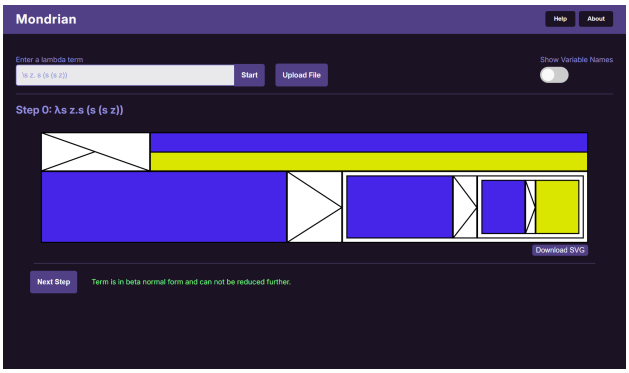


Figure 1: Screenshot of the Mondriλn+ web application.

Outlook

Mondriλn+ provides a strong foundation for mathematics and computer science education. Future enhancements could expand the application's capabilities. These include step-by-step animations that visualize how expressions simplify during computation, making the reduction process easier to follow or the ability to scan Mondriλn+ images directly from printed materials (similar to QR codes). This would allow students to seamlessly transition between paper and digital exploration.

Contents

1 Objectives

This section outlines the main objectives of the project.

1.1 Goals

The main goal was to turn the proof-of-concept implementation of Mondriλn into an interactive web application (that will be referred to as Mondriλn+ for simplicity) that should help beginners understand the concepts of the lambda calculus by providing an interactive and fun way of exploring and experimenting with lambda terms. Users should be able to create, export and import Mondriλn+ programs. Further, users should be able to perform step-by-step reductions on Mondriλn+ programs and modify their graphical representations. More specific, the individual panels within a Mondriλn+ program can be resized by the user and colors used for each variable can be replaced (α-conversion). Additionally, the language specification should be extended to support syntax conventions which simplify the visual representation of lambda terms.

1.2 Constraints

The web application has to be implemented in Haskell and should use a Haskell web framework.

1.3 Deliverables

The following deliverables were defined:

- Static web application, that doesn't require a backend.
- Detailed product documentation in form of a Markdown Wiki, to aid further development of the results.
- Project documentation, that only contains the information relevant for the current project.

2 Project Management

To manage the project we are using RUP for long term planning. We decided to use RUP since we both have already used it in previous projects and have made good experiences with this methodology. For short term planning we are using Jiras Kanban board feature. We are not facilitating scrum routines like daily standups or sprint retrospectives, since that would be to much overhead for a team of only two people.

2.1 Project Plan

The project plan was created using Jiras timeline feature. In Figure 2, the blue bars indicate the RUP phases, bright green markers denote completed milestones and dark green markers show milestones that were not completed.

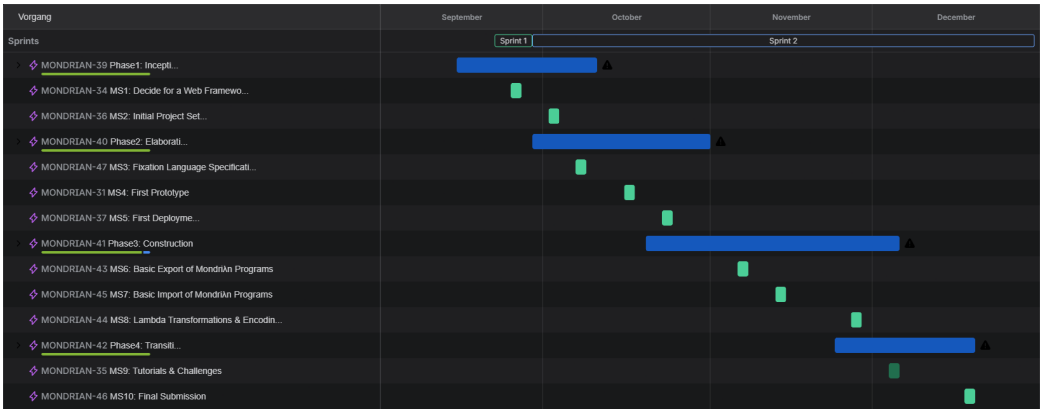


Figure 2: RUP project plan

2.2 Risk Analysis

We have identified the following risks that could potentially harm the execution of the project.

1. Unplanned absence of a team member
2. Time management
3. Missing the core goals of the project

4. Lacking Haskell and/or lambda calculus knowledge
5. Inexperience with Haskell web frameworks
6. Breaking changes within Haskell-Miso

2.2.1 Risk assessment

We have assessed each risk based on likelihood and impact, as can be seen in Table 1. Both likelihood and impact can be classified as either **High**, **Medium** or **Low**.

Risk	**Likelihood**	**Impact**
1. Unplanned absence of a team member	High	Low
2. Time management	Medium	Medium
3. Missing the core goals of the project	Medium	High
4. Haskell and/or lambda calculus knowledge	Low	High
5. Inexperience with Haskell web frameworks	High	High
6. Breaking changes within Haskell-Miso	High	High

Table 1: Risk assessment

2.2.2 Risk mitigation

Unplanned absence of a team member: It is very likely that one or both team members are absent for an unspecified period of time due to illness, an accident or other unforeseeable occurrences. However, the impact of this risk is low, since we already use tools like GitLab or Jira Cloud that help us collaborate and overcome such situations. In addition, we have already collaborated on previous projects and therefore, have already established channels that allow us to communicate outside of working hours. Thus, this risk has already been mitigated.

Time management: Struggling to manage our time appropriately is likely and has a medium impact. Our project planning using Jira helps us to reduce the likelihood of this risk occurring. We have reduced the likelihood of this risk further by reserving wednesdays and fridays for working exclusively on this project. With this reduction of likelihood the risk has ben mitigated as far as possible.

Missing the core goals of the project: This project is all about improving an existing application. Since "improving" is a vague term that can be interpreted differently it is somewhat likely that we might end up missing the core goals of the project. We actively try to mitigate this risk by getting familiar with the existing application and how it could be improved, by elaborating meaningful requirements, by designing mockups and by having regular meetings and discussions with our advisor.

Haskell and/or lambda calculus knowledge: The impact of having to little knowledge on Haskell and/or the lambda calculus would have a high impact on this project. However, the likelihood of this being the case is rather low, since we both have already invested time in refreshing our Haskell and lambda calculus knowledge.

Inexperience with Haskell web frameworks: Both team members have never before used Haskell web frameworks to develop web pages. Hence, the risk of struggling with the usage of a framework is very likely. With the impact also being high, it is the most important risk to mitigate. We are trying to mitigate this risk by conducting research on different frameworks and test them in order to choose the one that we both feel the most comfortable with.

Breaking changes within the Haskell Miso framework: Within the course of the project Haskell Miso was selected as the web framework. The project encountered two separate breaking changes introduced by active upstream development in Miso, which caused our code to no longer compile. To completely eliminate further disruption, the version was freezed to a specific commit hash.

2.3 Tools

Working on this project requires us to use some third party tools in order to improve efficiency and quality.

2.3.1 Jira

As already mentioned at the beginning of this chapter, we are using Jira to manage our work and track our issues as well as milestones and long term

timeline. We are also facilitating Jiras Kanban and time sheets functionality in order to generate time tracking reports.

2.3.2 Git and GitLab

For version control we will be using Git with GitLab as remote repository.

2.3.3 LaTeX

We are using LaTeX to write the project documentation.

2.3.4 Markdown and GitLab Wiki

To write the product documentation in form of a Wiki we are using Markdown and the GitLab Wiki feature.

2.3.5 Copilot and ChatGPT

In order to work as efficient as possible we are using generative artificial intelligence tools like Copilot and ChatGPT. We use these tools as a help for research, debugging and code generation. Every factual information generated by these tools is manually double checked using independent sources. We are not using any AI tools to generate documentation and we are actively making sure that our AI usage complies with the "Leitlinie zum Umgang mit KI-basierten Hilfsmitteln in Lehre und Weiterbildung" [1] and the "Umgang mit KI bei schriftlichen Arbeiten" [2] of the "OST - Ostschweizer Fachhochschule".

2.4 Quality Measures

In order to achieve the expected and also our own quality standards we have defined some quality guidelines that we will follow.

2.4.1 Git strategy

We do not think that a specific git strategy is necessary, since our team consists of only two people. However, we have agreed to not commit to the main branch and to rebase our feature branches when it makes sense in order to keep a forward flowing git history.

2.4.2 Commit messages

Commit messages should be short and meaningful. We will not expect from each other to strictly follow the specification of conventional commits [3], however, commit messages should be in a similar manner whenever it makes sense.

2.4.3 Merge requests

Every merge to the main branch has to undergo a review in form of a merge request. The review process is to ensure that any documentation satisfies the documentation quality measures and that any code satisfies the code quality measures. Further, the review process makes it less likely to push erroneous code or bugs to the main branch.

2.4.4 Definition of done

Any change, task or feature must fulfill the following requirements in order to be considered done:

1. The changes have led to the expected and intended behavior
2. All tests succeed
3. The changes have been reviewed in form of a merge request
4. The changes have been merged into the main branch
5. The changes have been documented

2.4.5 Code

A hard requirement is for any written code to be understandable by someone who has completed an introductory course in functional programming. Further, the code should be appropriate to be used as a showcase for programming in the functional style. To achieve this, we will stick to the following clean code [4] guidelines:

1. Names should be meaningful
2. Functions should only do one thing

3. Code should only be commented when it it makes sense and is necessary
4. Code should be formatted, such that it reads like a news paper
5. Code should be checked in cleaner than what it was checked out
6. Clearly separate pure from impure logic when dealing with side effects
7. Stick to common and domain specific conventions instead of trying to invent new conventions

2.4.6 Documentation

Any documentation has to be written in correct and understandable english. All team members are required to use a spell checker extension of their choice whenever writing or editing any documentation. All project related documentation has to be written down in this document, while all product related documentation has to be written down in the GitLab Wiki. All documentation has to be up to date with the current state of the project at all times.

3 Foundations and Borrowed Components

Rather than starting from scratch, this project builds on multiple existing foundations, integrating established concepts and proven techniques from prior work while advancing them to meet new requirements.

3.1 Mondriλn

The initial Mondriλn prototype stems from a predecessor thesis, that established the core concept of visualizing lambda calculus as abstract diagrams. We adopted and refined the visual language defined in that work, enhancing it to better represent complex expressions and improve usability.

Further, we inherited the automatic color selection algorithm that generates random high-contrast colors by leveraging HSB color space (Hue, Saturation, Brightness), ensuring visually distinct differentiation of variables while maintaining aesthetic consistency. [5]

3.2 lambdacalc.dev

Both the core lambda term structure and the overlay-menu component were adapted from the `lambdacalc.dev` source code. Another OST-based project offering an interactive lambda calculus calculator. [6].

4 Personal reports

This chapter contains the individual reports of each member after project completion.

4.1 Report by Sven Elvedi

This SA was a great opportunity to improve my functional programming skills, particularly in Haskell. Before this SA pretty much all my Haskell knowledge came from the FP course in the 2nd semester. FP obviously was a great course, but since there is no FP 2 or FP Advanced you really only get an introductory level of knowledge. I think that after the FP course many students (including me) are left with questions like "Do people actually use Haskell for non educational purposes?" or "How would you even write a UI in Haskell?". This SA has clearly answered those question for me. I've really gotten to experience the strengths (type safety, purity, immutability) and weaknesses (steep learning curve, tooling complexity, framework maturity) of Haskell and learned that writing Web UIs in Haskell is actually pretty enjoyable. I also really enjoyed using Miso for this project. With every feature I implemented I started to like Miso, and generally the Elm architecture, more and more. One unfortunate drawback of Miso is that it is, for the most part, maintained by one single person (possibly in his/her free time). We did have to deal with one breaking change in the Miso repository, which required us to fix the dependency to a specific commit hash in order to not have to lose too much time refactoring our code to work with the new Miso version. But apart from that the Miso maintainer is really doing an impressive job with maintaining the framework.

When I now look at what we achieved during this SA it feels a bit underwhelming at first, because the final product is "just" a small web app that visualizes lambda calculus expressions in contrast to a full stack web application or a complex backend service. When you look at the time you have for an SA it is definitely possible to build more complex systems using a programming language, architecture, framework and patterns that you are already familiar with. But for me this SA was not about building the most complex system possible, it was about learning new things and getting out of my comfort zone by using a programming language, a framework, an architecture and patterns that I am not really familiar with. Of course I would have liked to implement some more features, like making the SVGs more

interactive than just dragging the edges or animations for the reduction and conversion processes.

Some further takeaways (mainly things that I think we did not do that well) from this SA that will hopefully prepare me for the BA and future projects are:

- Don't spend too much time on small details that are not important for the overall goal of the project.
- Talk as much as possible to your teammates and advisors to make sure that you are on the right track and that you don't work against each other or simultaneously on the same thing.
- Show your progress and ask for feedback as often as possible.
- Ask questions as soon as you have them, don't wait until they pile up. Even "stupid" questions can often lead to important insights.
- Define stricter code guidelines at the beginning of the project to avoid unnecessary discussions about code style and structure later on.

Overall I am happy with what I achieved and with what I learned during this project and I think it gave me a lot of valuable experience for the BA and other future projects.

4.2 Report by Timo Strolz

Like Sven, I initially doubted whether Haskell could be used to build real world applications, that work with user inputs, as well as even creating a web application that runs within the browser. This was the main question I set out to answer at the start of this project. My conclusion: Haskell is indeed practical for real applications, and using it was an enjoyable experience. Miso turned out to be really pleasant to work with, though it is not quite mature in some areas. It's simple, straightforward, and I appreciate its architecture. Working with Haskell in general presented its own challenges, documentation is often very sparse and the examples are often overly abstract or mathematical. But over the course of the project I grew to appreciate Haskell: once the code compiles and runs it behaves predictably and rarely produces surprising side effects. By contrast, JavaScript often appears to work, but frequently leads to unexpected behavior. Later in the

project I discovered the "Learn You a Haskell" book, which proved an to be a valuable resource, it covers a lot of Haskell topics, and provides clear and practical explanations. I also learned the power of breaking problems into smaller, manageable parts. What initially seemed daunting, often became straightforward once decomposed. Perhaps most importantly, I discovered that sometimes, its best to step away from the keyboard and really think through a problem first. In fact, one particularly problem, finding all related variables for alpha-conversion, was actually solved during a train ride and not in front of a computer.

What I would do differently next time:

- Start working on documentation earlier. I noticed that motivation increases as the deadline approaches, and I would have preferred to channel that energy into coding rather than writing documentation.
- Establish a clean CSS and HTML structure from the beginning, taking responsiveness into account from the start to avoid structural refactoring later.
- Do more user testing. We built an interactive web application that is meant to be used by people, but we only had a few people test it out.
- Refer back to the initial project plan regularly, we did it at the start but seldom consulted it afterwards. Kind of felt like an artifact, that we only created for the documentation.

Personal Highlights:

- Got to work with cool technologies like Nix and WebAssembly.
- I also had the opportunity to explore another non–JavaScript web framework (last semester it was Blazor with C#).
- Learned a lot about functional programming in Haskell. As well as its approaches to solving problems.
- Recursive structures in Haskell are remarkably robust. Once something works, it consistently works without unexpected side effects.

References

[1] Kommission Lehre und Wieterbildung (2024). *Leitlinie zum Umgang mit KI-basierten Hilfsmitteln in Lehre und Weiterbildung*. OST - Ostschweizer Fachhochschule. [https://ostch.sharepoint.com/teams/TS-StudiengangInformatik/Freigegebene%20Dokumente/Studieninformationen/Leitlinie%20KI-Einsatz%20in%20Lehre%20und%20Weiterbildung.pdf] (visited on 24th September 2025)

[2] Departement Informatik (2024). *Umgang mit KI bei schriftlichen Arbeiten*. OST - Ostschweizer Fachhochschule. [https://ostch.sharepoint.com/teams/TS-StudiengangInformatik/Freigegebene%20Dokumente/Studieninformationen/D-I-Umgang-mit-KI-bei-schriftlichen-Arbeiten.pdf] (visited on 24th September 2025)

[3] Conventional Commits contributors (2025). *Specification*. Conventional Commits. [https://www.conventionalcommits.org/en/v1.0.0/#specification] (visited on 26th September 2025)

[4] Martin, Robert C. (2009). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR

[5] Friedrich, Anja and Panchaud, Mona (2024). *Mondriλn: A Visual Programming Language Based on the Lambda Calculus*. [https://eprints.ost.ch/id/eprint/1264/]

[6] Buchli, Lukas and Prof. Dr. Mehta, Farhad D. (2022). *lambdacalc.dev: Lambda calculus calculator*. [https://lambdacalc.dev] [https://gitlab.com/LambdaCalc/LambdaCalc]

List of Figures

List of Tables

Appendix

A Meeting notes

A.1 Meeting 27.05.2025

- Kick-off meeting
- Talk about different ideas for a semester thesis

A.2 Meeting 02.09.2025

- Decision to do our semester thesis on the Mondriλn project.

A.3 Meeting 17.09.2025

- Review of the task description

A.4 Meeting 24.09.2025

- Finalization of the task description
- Review of our current project plan and timeline

Key take aways:

- Definition of a milestone for the language specification
- Further analysis of the current state of the Mondriλn project

A.5 Meeting 01.10.2025

- Mondriλn+ encodings
- Making the encoding of λ w x y z . w x y z look better (Syntax conventions)
- Graphical interface of the end product.

Key take aways:

- New approaches for encoding of bound variables discussed. (Abbreviated syntax, all combined in one square)

- Brackets could be encoded as a white colored frame. (Other color would also be possible...)
- The two tabs in the current UI sketch (generate & parse) could be combined into one.

A.6 Meeting 08.10.2025

- Further discussions about encodings and representations
- Decisions regarding language specification

Key take aways:

- Finalize language specification and implement first prototype

A.7 Meeting 22.10.2025

- Presentation of current progress
- Discussions about what to do next
- GitLab Wiki

Key take aways:

- Start working on SVG rendering to find out what is possible (POC)
- Rather implement things from scratch than spending to much time on trying to integrate existing Code
- Since the new application is different from the old console-app, the decisions made in the predecessor-sa should not influence our code-base.
- Stick with GitLab Wiki for product documentation to see how it goes
- Define use cases and requirements to be complete in themselves instead of referencing the initial Mondriλn project
- Use cases should help us complete the project, not generate overhead

A.8 Meeting 29.10.2025

- Presentation of current progress
- Discussions about SVG-POC
- Discussion about SVG handling and how to represent SVGs in the model
- Discussion about reevaluation of Reflex-FRP in regards to SVG support

Key take aways:

- SVG size should be stored in ratios (similar to screen ratios i.e. 16:9)
- A reflex poc should be done
- Come up with a datatype that combines lambda terms and their SVG representations
- Docs will be handed in before the next meeting

A.9 Meeting 05.11.2025

- Current state of the docs was handed in, seems okay so far.
- Discussion Miso vs Reflex
  - Reflex is not really better suited for handling dynamic SVG's, we plan to stick with Miso.
- Discussion about current and future problems with SVG editing
  - Ask Lukas Buchli for help, since he has experience with Miso and possibly Reflex.
  - Use an intermediate representation.
  - Ask within the haskell discord/reddit
  - Use relative sizing
  - Horizontal and vertical application should be possible.
- Discussion Mondriλn+ datatype

  - Tracking between the actions performed on the site (e.g. clicking) and the concerned part of the Mondriλn+ term should be done via the cursor position.
  - Recursive approach: Calculate relative coordinates of current target, step into nested parts and so on.
  - Reference into ADT's → Record the path taken within the data structure → Done within LambdaCalculator.
  - Relative size is always 1, sub-parts add to 1, also recursive.
  - Another mentioned approach were zipper datatypes.
- Recommendations for Testing
  - Use hUnit.
  - Do as much property-based testing as possible.
  - Do at least some unit-tests.
  - The rest needs to be tested manually.

A.10 Meeting 12.11.2025

- Short meeting
- Nothing specific to report
- Quick chat about time management

A.11 Meeting 19.11.2025

- Again a short meeting
- Nothing specific to report
- Quick demo of the current state

A.12 Meeting 26.11.2025

- Discussion about two different SVG structures
  - First structural approach keeps the SVG as simple as possible and does not encode any specific information about the lambda term.
  - Second structural approach builds the SVG as a very hierarchical XML tree that encodes the recursive structure of the lambda term.
  - We prefer the second approach, but we still wanted to talk about the two possibilities.
- Demo of current state
- Discussion about next steps, what lies withing the realistic scope and where to place the focus on.

Key take aways:

- Stick with the hierarchical SVG structure to allow easy parsing and interactivity while accepting the drawback of not being able to parse SVGs that might look the same but don't follow that structure strictly.
- What we decided on trying to finalize until the end of the project:
  - Hide/show variable names, the lambda symbols in the squares.
  - Modify graphical representation: α-conversion / change color and resizing parts of Mondriλn+ programs.
  - Improve visual appeal.
  - Product and project documentation has priority over the paper (technical report in ACM Sigplan form).

A.13 Meeting 03.12.2025

- Short meeting
- Nothing specific to report
- Unfortunately we could not show a demo because we had some Miso dependency issues, which we solved after the meeting.

A.14 Meeting 11.12.2025

- Demo of current state
- Discussion about open features
- Discussion about strategy for the last week → What has to be done and what not
- Ask open questions

Key take aways:

- Focus on docs and a clean code base instead of rushed features
- Only show the newest term on the site, remove the scrolling
- If a previous term in the history is edited, a new future-path is taken, the history changes from there → other steps from there get lost
- Text-toggle should only remove variable-names, the lambdas and arrows should always be displayed
- Currying syntax → Update the term structure for abs into [var] (min 1 var) and remove the toggle → currying is controlled by the chosen syntax
- Brackets → could be embedded into the AST-Structure, support the recursive nature
- α-conversion → Prohibit shadowing and make suggestions for matching colors.
- Prevent functions with loads of parameters → Introduce records and split up the functions.
- Rotational invariance → Talked about, but will most likely not be implemented.

A.15 Meeting 17.12.2025

- Review of the abstract
- Discussion about AST/CST modifications.
- Discuss the required submission format for the hand-in on friday.

Key take aways:

- The background color of the website should be made a bit brighter to improve visibility.
- Export the gitlab wiki to pdf and upload it to AVT.
- Stick with the current AST structure for now, the infos for the new syntax conventions are only stored within the VisualInfo nodes, the AnnotatedTerm structure remains unchanged. Making such drastic structural changes within the final three days would be too time-consuming and risky.
- Abstract
  - The title should still be Mondriλn and not Mondriλn+, although we can refer to our version as Mondriλn+ in the documentation.
  - Don't use the word flaws when talking about the previous version, rather focus on the improvements that were made.
  - Miso documentation: It is not broad, but it is existing. (Which is not that common within the Haskell ecosystem.)

B Time tracking reports

B.1 Time spend by person

Figure 3 shows the time spent by each team member during the project.

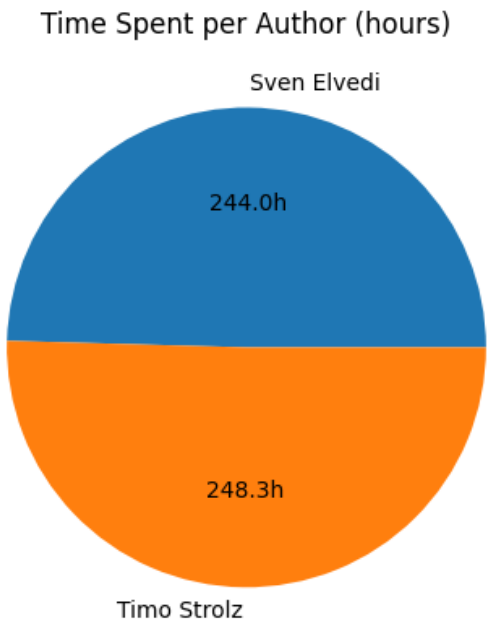


Figure 3: Time spend by person

B.2 Time spend by label

Figure 4 shows the time spent on different project activities, categorized by labels.

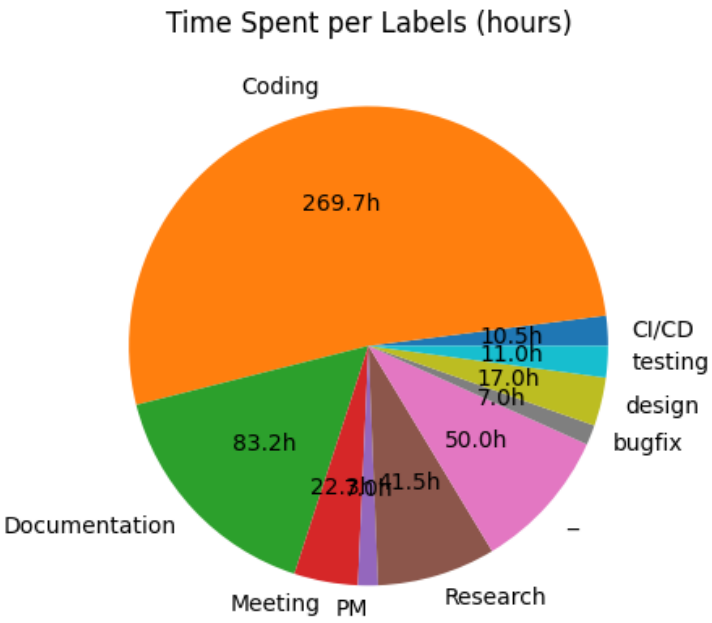


Figure 4: Time spend by label

C Task Description

Improving the Usability of MondriΛn, a Visual Esoteric Functional Programming Language

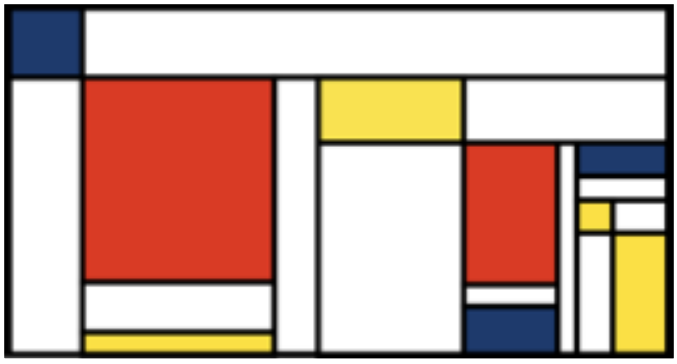
Task Description

1. Setting

An esoteric programming language (esolang) is a programming language designed to test the boundaries of computer programming language design, as a proof of concept, or as software art[1]. Although not used for working developers to write software, esolangs allow its users and developers to experiment with and sometimes even learn programming language styles and features in an easy, minimal and often enjoyable way. The nonstandard constructs of esolangs force its users to think about programming and problem-solving in new and innovative ways. Designing and implementing esolangs allow one to gain first-hand experience in several computer-science disciplines, formal and practical, such as grammar rules, lexical analysis, parsing, run-time systems, and Turing completeness in a relatively short amount of time.

Most of the esolangs in (dis)use today are imperative (i.e., where programs are statements that change program state), and explicitly designed to be a challenge to use. The design space of functional esolangs (i.e., where programs are expressions that are evaluated using equational rewriting) seems to be less well explored.

A prior project[2] resulted in the conception and development of a proof-of-concept implementation of MondriΛn, a visual programming language based on the lambda calculus.



[1] https://en.wikipedia.org/wiki/Esoteric_programming_language
[2] https://eprints.ost.ch/id/eprint/1264/

2. Goals

The main aim of this project is to improve the usability of Mondriλn by improving its conceptual design and constructing software tools that allow users to comfortably experiment with and explore the design space offered by this language.

The following is a brief and unstructured list of initial tasks, requirements, and notes:

1. The conceptual design of Mondriλn should be extended to support visual counterparts of syntax conventions for multiple bounding occurrences and multiple applications.
2. The design and implement a static web application (i.e. that needs no server back-end) written in Haskell that
    a. Allows users to generate Mondriλn programs (visual representations of lambda terms in the Mondriλn format) from lambda terms and visa versa.
    b. Allows users to import and export Mondriλn programs and their corresponding lambda representations.
    c. Allows users to construct Mondriλn programs and their corresponding lambda representations.
    d. Allows users to modify the graphical representation of Mondriλn programs, for instance by resizing subprograms or colour replacement (the Mondriλn equivalent of α conversion).
    e. Allows users to perform step-by-step reduction on Mondriλn programs.
3. The main aim of the application is to help beginners understand the lambda calculus in an enjoyable way, using an alternative approach.
4. Special care should be taken to make the application usable and visually appealing.
5. The application should be used to explore and document the Mondriλn equivalents of common transformations and encodings in the lambda calculus, for instance the Church encodings, Scott encodings, the Y Combinator, point-free transformation and various combinatory transformations.
6. A series of tutorials or challenges should be provided, at best in the application itself, that allow users to learn how to use the application and to improve their problem-solving skills.
7. Further points to consider could, if time and interest allow, include:
    a. Being able to generate a Mondriλn program from a bitmap image, so that it can be used like a Q-R code.
    b. Extending Mondriλn to terms with free variables (including constants).
    c. Exploring the design space of representing any free algebraic data type graphically.
8. The implementation code should be understandable by someone who has completed an introductory course in functional programming and be appropriate to be used as a showcase for programming in the functional style for such people.
9. The results of this project should be documented in a manner that entices its immediate use.
10. The results will be released under the GPLv3 licence.

Due to the explorativeitative nature of this project, further refinements and modifications to this list that serve the main aim of this project are possible during its course.

Although the current task description is centred around the visual programming language Mondriλn, it would also be possible to centre the project around an alternative, possibly still to be discovered, visual functional programming language.

3. Deliverables

- All artifacts (source code, web pages, etc.) required to achieve the goals of this project.
- Product documentation in English that is relevant to the use and further development of the results (e.g., requirements, domain model, architecture description, code documentation, user manuals, etc.) in a form that can be developed further and is amenable to version control (e.g., LaTeX or Markdown). Ideally, all product documentation should be contained within the artifacts required to achieve the goals of this project as stated in above.
- A technical report in the form of an ACM SIGPLAN paper that describes the conceptual and technical results of this project.
- Project documentation that is separate from the product documentation that briefly but precisely documents information that is only relevant to the current project (e.g., project plan, time reports, meeting minutes, personal statements, etc.).
- Additional documents as required by the department (e.g., poster, abstract, presentation, etc.)
- Any other artifacts created during the execution of this project.

All deliverables may be submitted in digital form.

4. Stakeholders

Industry Partner: None
Students: Sven Elvedi, Timo Strolz
Supervisor: Farhad Mehta.

5. Other Project Details

Type of project: Study Project (de: Studienarbeit)
Duration: 15.09.2025 – 20.12.2024
Workload per student: 8 ECTS (1 ECTS = approx. 30 Hours)
Grading: Organization and Execution 20%; Formal Quality of the Report 20%;
Analysis, Design & Evaluation 20%; Technical Realization 40%.

D README

Improving the Usability of Mondriλn, a Visual Esoteric Functional Programming Language

Quicklinks

Live Version of Mondriλn deployed on GitLab Pages

Product Documentation on GitLab Wiki

Project Documentation deployed on GitLab Pages

Test coverage report deployed on GitLab Pages

Usage

Mondriλn makes learning, teaching and just playing around with the lambda calculus fun and visually appealing.
You can enter lambda calculus terms and generate SVGs that represent the entered term.
The generated SVGs can be customized in the following ways:

- Resize individual elements of the SVGs by dragging the edges that separate the elements.
- Change colours of elements (in an α-conversion like way) by selecting them.
- Toggle between showing or hiding variable names in the SVG.

Once a valid term has been parsed, you can click on "Reduce" to β-reduce the terms and images step by step and visually inspect what happens at every reduction step.
Download any displayed SVG at any point of the reduction process to upload it later and continue where you left of, to share them with friends and colleagues or to decorate your walls with them 😉

Give Mondriλn a try!

Development

The app is written in Hakell using the Miso framework and Cabal in combination with Nix as built tool.
To run the app locally enter a nix shell in the projects root directory:

```
nix develop --experimental-features nix-command --extra-experimental-features flakes
```

Once inside a nix shell you can build the app with:

```
cabal build
```

Developing is most comfortable by entering the ghci shell with:

```
cabal repl app
```

and then calling the main function to host the app on localhost:8008:

```
main
```

Tests can be run with:

```
cabal test
```

A test coverage report automatically gets deployed by the pipeline and is located here.

Deployment

The GitLab CI/CD pipeline compiles the app to WASM and deploys it to GitLab Pages.
Preceding pipeline jobs run a formatter check and execute all test cases.

The pipeline also deploys the project documentation and the test coverage report to GitLab pages.

Formatter

The app uses fourmolu to format the source code.
The pipeline fails if the code is not formatted as described in the fourmolu config file.
The repository contains a pre-commit hook that can be placed inside the .git directory of your local repository to run a formatter check before every commit.

Documentation

The project documentation gets deployed by the CI/CD pipeline to GitLab pages and is accessible here.

The product documentation is a GitLab wiki located here.

GitLab handles Wikis as separate repositories with their own history and version control.

You can clone the wiki repository with:

```
git clone ssh://git@gitlab.ost.ch:45022/MeFa_Proj/2025-HS_SA_Mondrian/mondri-n.wiki.git
```

or

```
git clone https://gitlab.ost.ch/MeFa_Proj/2025-HS_SA_Mondrian/mondri-n.wiki.git
```

> Note that GitLab uses a slightly modified version of Markdown, so your mainstream Markdown editor might struggle rendering certain things like cross references or image captions. We suggest browsing and editing the Wiki on GitLab instead of locally.

E Product Documentation (GitLab Wiki)

E.1 Home

Mondriλn [1] is a proof of concept for an esoteric functional programming language [2] based on the lambda calculus. The goal of this project is to extend both the functionality and usability of Mondriλn.

Mondriλn

The current version of Mondriλn is a CLI tool implementing the following use cases:

1. Convert a lambda term to an SVG
2. Beta reduce a lambda term and convert it to an SVG
3. Convert a Mondriλn SVG to a lambda term
4. Convert a Mondriλn SVG to a beta reduced lambda term

The following figures depict some examples for the resulting Mondriλn SVGs.

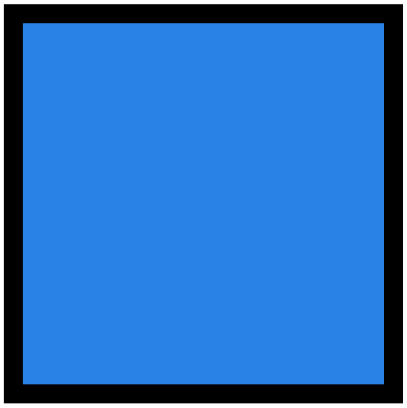


Figure 0.1: Mondriλn SVG for the variable ***x***

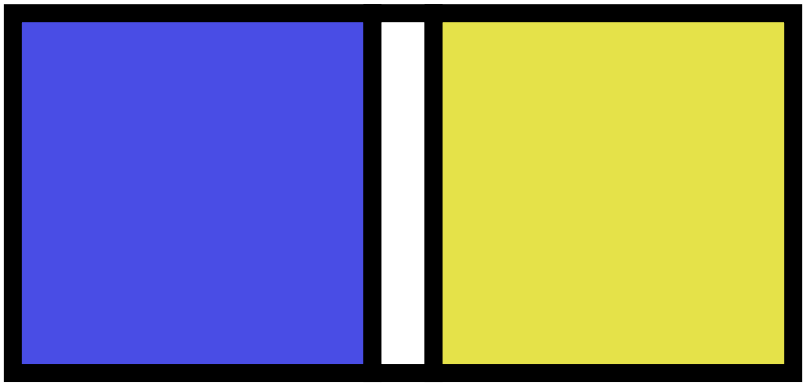


Figure 0.2: Mondriλn SVG for the application ***f x***

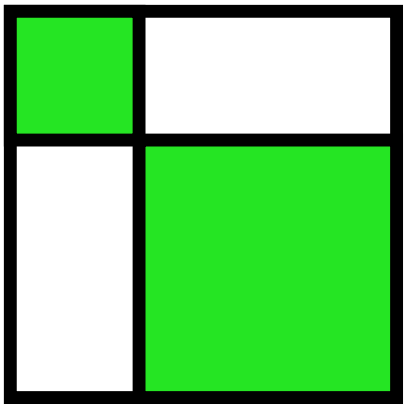


Figure 0.3: Mondriλn SVG for the abstraction ***λx. x***

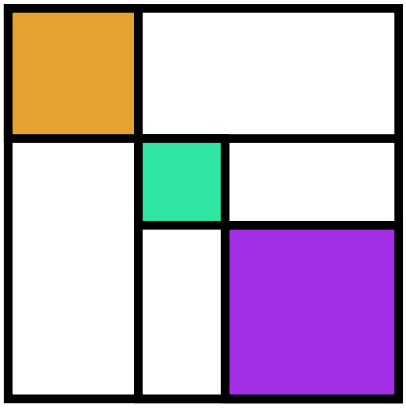


*Figure 0.4: Mondriλn SVG for the nested abstraction **λx. λy. body***

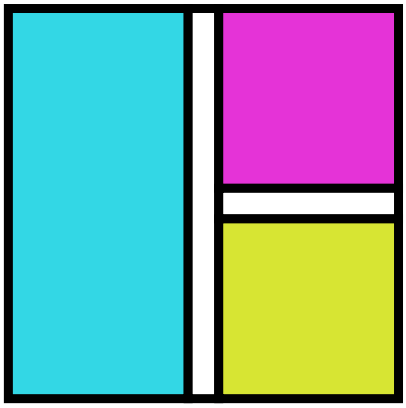


*Figure 0.5: Mondriλn SVG for the parenthesized application **f (g x)***

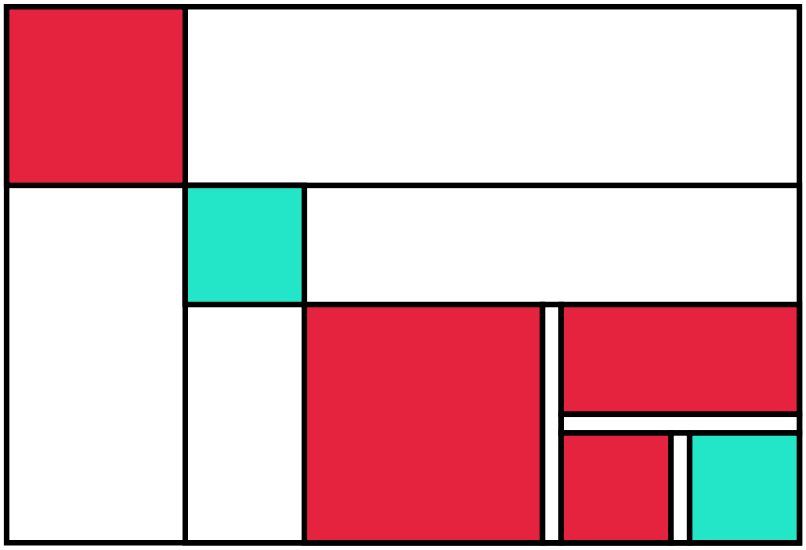


*Figure 0.6: Mondriλn SVG for the for the Church numeral 3 **λs. λz. s (s (s z))***

The current version of Mondriλn is a good proof of concept. However, since it is only a proof of concept, it does have some room for improvement.

Areas of improvement

While using and getting familiar with Mondriλn we noticed the following aspects that could be improved.

Usability

Being a CLI application, its use is limited to people that know how to use CLI tools. Further, the interactivity achievable with a CLI-tool is also limited.

β-reduction

The current process of β-reduction is somewhat laborious, since you have to feed each, by one step reduced, representation back to the application in order to reduce it one step further.

α-conversion

The current version of Mondriλn does not implement any functionality for α-conversion equivalent transformations on the SVG images.

Visual representation of certain lambda terms

Certain lambda terms produce visually unappealing Mondriλn representations. For example, terms with multiple bound variables result in long diagonals, as can be seen in Figure 0.4. Other issues include:

- Parenthesized applications are hard to reconstruct from the Mondriλn representations (as can be seen in Figure 0.5)
- Poor scaling for more complex lambda terms
- No customizability for resulting images

Improvements

This new version of Mondriλn, that will be referred to as **Mondriλn+** for simplicity, implements the following improvements.

A web application instead of a CLI tool

Converting the command-line tool to a web application significantly improves usability. Users no longer need to install or run code locally; instead, they access an intuitive, visually designed interface that requires no familiarity with command-line syntax or external documentation.

Changes to the language specification for Mondriλn representations

The Mondriλn language specification has has been reworked in order to implement the discussed improvements. The latest version of the entire language specification can be found under Language specification.

Abbreviation of lambda terms with multiple bound variables

Higher order lambda terms (i.e. with multiple bound variables) often get abbreviated in a style, where only one lambda is written down, instead of writing down every lambda. For example, the higher order lambda term **λa. λb. λc. λd. body** abbreviated in this style results in **λa b c d. body** [3]. The initial version of Mondrivλn can not handle such abbreviations. However, the new Language specification, as well as the parsers, can handle such abbreviations and encode them into the generated representations, giving the user the choice of which syntax to use.

Representation of parenthesized applications and order of application

The new Language specification encodes explicit parenthesis by using white boxes and makes the order of application explicit by using arrow-like lines pointing in the direction of the application order.

Manual visual modifications

The user has the ability to manually modify the visual representation of a term. For example by changing the fractions that each element of the SVG takes up or by changing the color of variables, which is equivalent to an α-conversion.

Step by step β-reduction

The current version of Mondriλn does implement step by step β-reduction functionality. However, the user manually has to re-parse the reduced lambda term or SVG in order to reduce it one step further. The web application, on the other hand, allows the user to enter a lambda term or parse an SVG and then reduce it one step at a time by just clicking on a button.

Domain analysis

Since this is an extension of an existing project rather than something completely new there was no need to perform an in depth domain analysis. Such an analysis has already been done in the initial version of the Mondriλn project [1]. The domain analysis, including minor additions and changes that were made to it,

can be found in Domain analysis.

Architecture

Architectural documentation and decisions can be found in Architecture.

Comments

E.2 Domain Analysis

The domain analysis is heavily influenced by the domain analysis of the existing MondriΛn project [1]. Use cases as well as functional and non-functional requirements have been taken over from the existing project and have been modified to adapt the changes made to the architecture, language specification and user experience of the application.

Actors

There is only one kind of actor for this system, therefore we employ the "User" as a primary actor for all use cases.

Use-Cases

Use cases are annotated with UC## in order to reference them easily.

UC01: Convert lambda term to image

The user inputs a valid lambda term on the website. The system converts the given term into an image in the defined visual representation format. This image then gets rendered on the website. If the user inputs an invalid lambda term the website shows an error.

UC02: Parse lambda term from image

The user uploads an image in the defined visual representation format on the website. The system parses the given image to receive the lambda term contained within. The lambda term is shown on the website. If the user uploads an invalid image the website shows an error.

UC03: β-reduce lambda term

The user has performed either UC01 or UC02 in the non erroneous scenario. The user can then click on the "Reduce" button in order to β-reduce both the lambda term and the image by one β-reduction step. The β-reduced lambda term and image get displayed beneath the original or previous term. The user can repeat this step until the term is in β-normal-form. Once the term is in β-normal-form the website displays an according message.

UC04: Edit image

The user has performed either UC01 or UC02 in the non erroneous scenario. The user can then edit the image in various ways that don't change the semantics of the term. Possible changes to the image are:

- Change colors (equivalent to α-conversion)
- Change size of image
- Change orientation of image
- Change aspect ratio of image

UC05: Download image

The user can download any generated image at any time during which the user performs UC01, UC02, UC03 or UC04.

Use case diagram

Figure 1.1 shows a basic use case diagram.

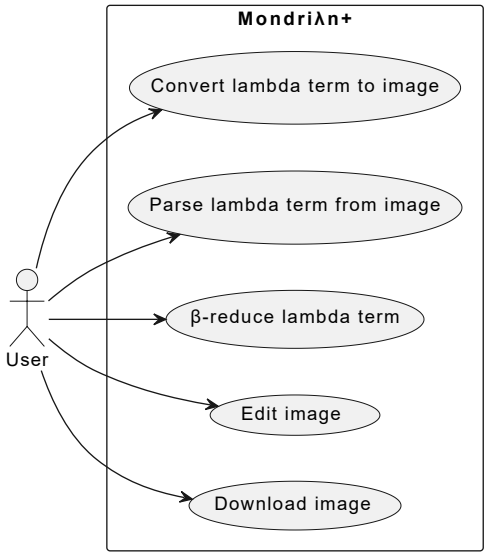


Figure 1.1: Use-Case diagram

Requirements

The requirements are structured according to FURPS [5].

Functionality

- RF01: The system is able to convert lambda terms containing any combination of variables, abstractions and applications.
- RF02: The system is able to convert lambda terms containing parentheses that alter the default order of application.
- RF03: The system is able to toggle and convert between curried and uncurried lambda terms.
- RF04: The system is able to detect an invalid lambda term and inform the user.
- RF05: The system is able to convert images containing any combination of variables, abstractions and applications.
- RF06: The system is able to convert images containing parentheses that alter the default order of application.
- RF07: The system is able to toggle and convert between curried and uncurried images.
- RF08: The system is able to detect an invalid image and inform the user.
- RF09: The system is able to β-reduce terms and images until they are in β-normal-form.
- RF10: The system uses the same color for a given variable name throughout the reduction process.
- RF11: The system applies alpha renaming to resolve shadowing when two distinct variables have the same color, assigning a new, unused color to one of them.
- RF12: The system is able to perform α-conversion on the terms and images according to the users instructions.

Usability

- RU01: Users who have had previous experience with the lambda calculus and have read the language specification for the images, are able to determine what lambda term is represented in images.
- RU02: Users are able to access documentation, tutorials and help on the website.

Reliability

- RR01: The system handles errors gracefully and provides users with a comprehensible error message.

Performance

- RP01: The system converts lambda terms to images within 3 seconds.
- RP02: The system parses lambda terms from images within 5 seconds.
- RP03: The system executes β-reduction on lambda terms in a rate of at most 5 seconds per reduction step (including the generation of images for each reduction step).

Supportability

- RS01: The web framework used for the application is open source.

- RS02: The source code, documentation and all project artifacts will be released under the GPLv3 license.

Website Flow

Figure 1.2 shows the application flow of the web application. Users can start by either entering a lambda term or uploading an image. If valid, the corresponding representation is calculated. Next, the term and image can be β-reduced in steps, meanwhile the user is able to customize the generated image at any reduction step.

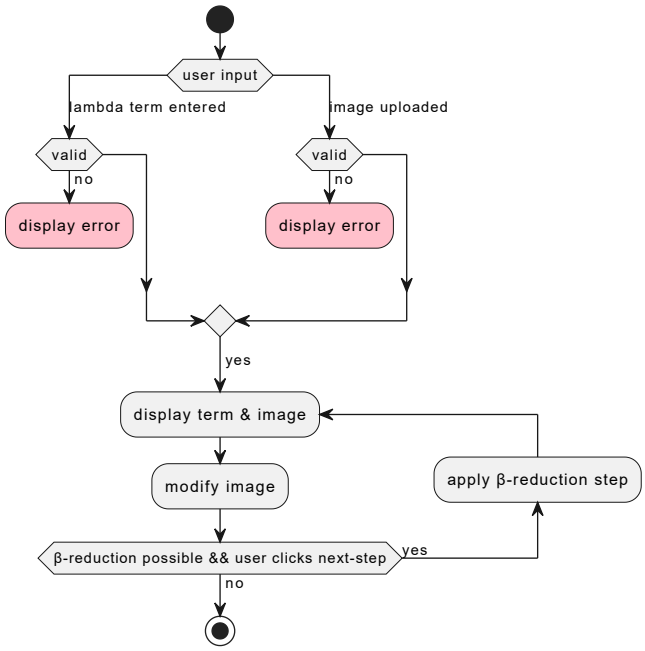


Figure 1.2: Activity-diagram of the website flow

Requirements Status

This section summarizes the implementation status of the project requirements. Each entry shows whether the requirement was reached or remains unresolved.

- RF01: Reached
- RF02: Reached
- RF03: Reached
- RF04: Reached
- RF05: Reached
- RF06: Reached
- RF07: Reached
- RF08: Reached
- RF09: Reached
- RF10: Reached
- RF11: Not implemented. Currently an error message is displayed in that specific case and the term is not parsed any further.
- RF12: Reached
- RU01: Reached. We tested the site with several classmates and received positive feedback.
- RU02: Not reached. Users are able to reach the doumentation via the overlay-menu and theres also a small help section there. But there are no tutorials provided on the website.
- RR01: Reached, although there is still room for improvement.
- RP01: Reached
- RP02: Reached
- RP03: Reached
- RS01: Reached
- RS02: Reached

Comments

E.3 Language Specification

On this page

The syntax of the pure lambda calculus is defined by the following grammar rule [4]:

$$M \quad ::= \quad x \quad | \quad \lambda x.\ M \quad | \quad M\ M$$

M: λ-terms

x: Variable (placeholder)

$\lambda x.\ M$: λ abstraction (anonymous function definition)

$M\ M$: Application of one λ term to another

Note: Since we are considering the pure lambda calculus, all identifiers belong to the syntactic category of variables.

$x, y, z,$
$a, b, c,$
$1, 2, 3,$
$+, -, *, \ldots$

Figure 2.1: *Lambda calculus grammar rule*

Further implicit rules are:

- The scope of a lambda abstraction extends as far to the right as possible
- Application is left associative
- Parenthesis can be used to overwrite the first two rules

Mondriλn+ representation rules

The following rules define how a lambda term is to be represented visually.

Variables

A variable is represented by a rectangle. The color of the rectangle represents the variable identifier.

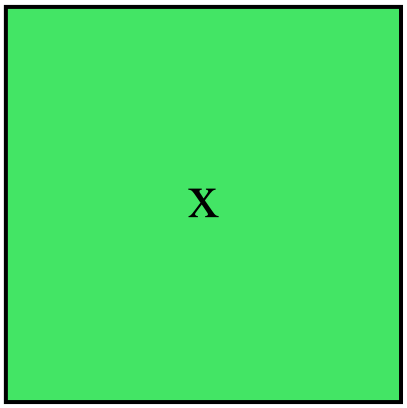


Figure 2.2: *Representation of a variable*

Abstractions

An abstraction is represented by a rectangle. A single line splits the rectangle horizontally in two halves. The top half is again vertically split in half by a single line. This results in one bottom rectangle and two top rectangles.

Single bound variable

In case of a single bound variable the top left rectangle contains one full and one half diagonal symbolizing a lambda. The top left rectangle is unique and can therefore be used to correctly orientate the abstraction. The top right rectangle contains the bound variable and the bottom rectangle contains the lambda term.

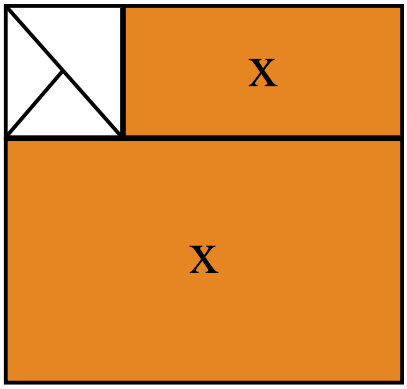


Figure 2.3: Representation of a single bound variable abstraction

Multiple bound variables

In case of multiple bound variables there are two scenarios.

Scenario 1: Uncurried

In case of uncurried bound variables the top left corner is the same as with a single bound variable. The top right corner is split either horizontally or vertically as many times as necessary to create the same amount of rectangles as bound variables.

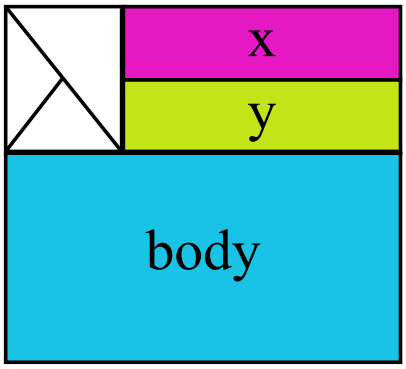


Figure 2.4: Representation of a multiple bound variables uncurried abstraction

Scenario 2: Curried

In case of curried bound variables, each nested abstraction (including its bound variable) is contained within the previous abstractions body. Therefore, each bound variable gets its own lambda.

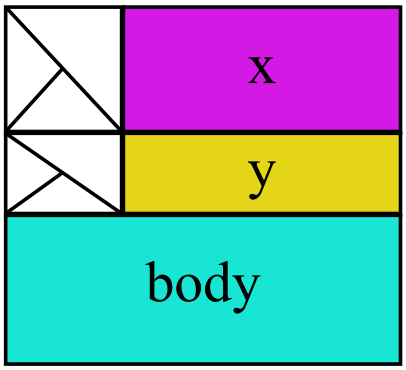


Figure 2.5: Representation of a multiple bound variables curried abstraction

Applications

An application is represented with a white space in between two terms. The white space contains two lines going from the two corners of one side of the white space to the center of the opposite site. These lines represent an arrow showing the order of application.

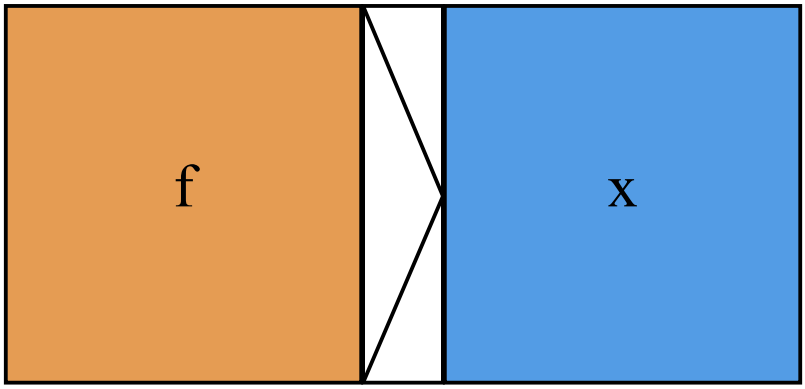


Figure 2.6: Representation of an application

Parenthesized terms

Parenthesis are represented by a white border around the parenthesized term.

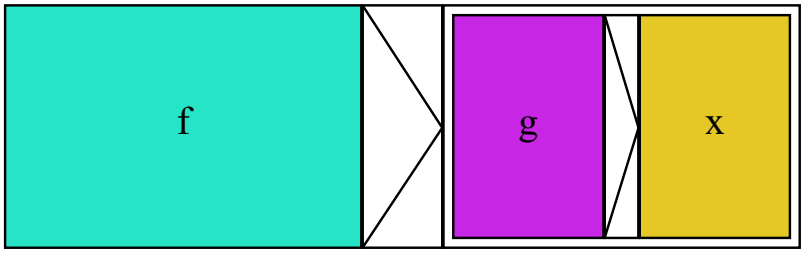


Figure 2.7: Representation of a parenthesized term

Specific example

As a specific example of a more complex term, Figure 2.8 depicts the generated image for the Church numeral 3 in the uncurried syntax style.

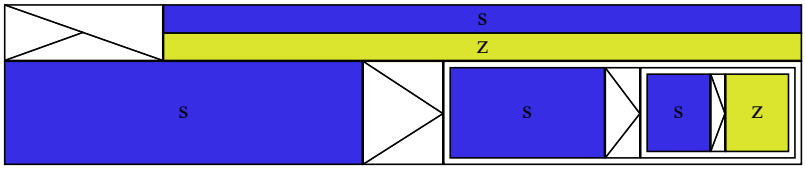


Figure 2.8: Representation of Church numeral 3

Unspecified aspects

Note how there are no specifications or restrictions on orientation, aspect ratio or size of the representations and that the specified rules also allow for rotational invariance.

Comments

E.4 Architecture

On this page

This page describes the architecture of the Mondriλn+ web application in form of a C4 diagram [10].

System Context

Figure 3.1 shows the system context of the application. Since Mondriλn+ is a standalone application it does not depend on any external software systems.

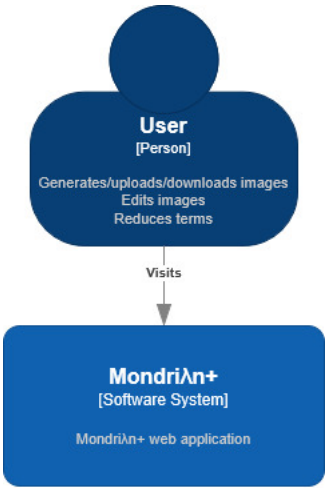


Figure 3.1: C4 system context diagram

Container View

Figure 3.2 shows a container view of the app. All Miso code is contained withing the web application container and all Miso independent domain logic gets imported from the library container.

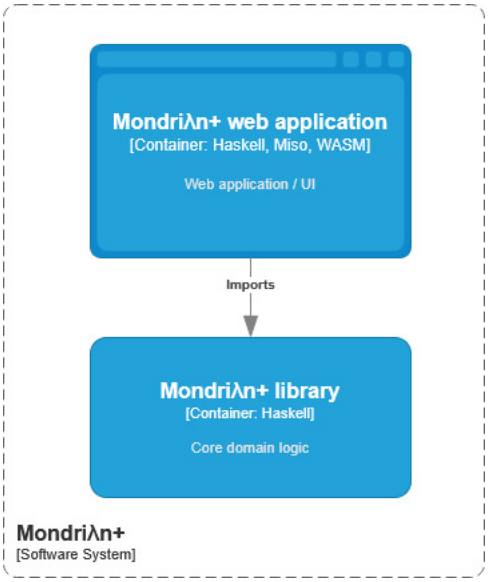


Figure 3.2: C4 container view diagram

App Component View

Figure 3.3 shows a component level view of the application. The depicted architecture is highly Miso specific, which in turn strongly follows the Elm architecture. SvgDragger, SvgRenderer and JsInterop have been implemented as separate modules for simplicity and separation of concerns but they also generally follow the Miso/Elm architecture.

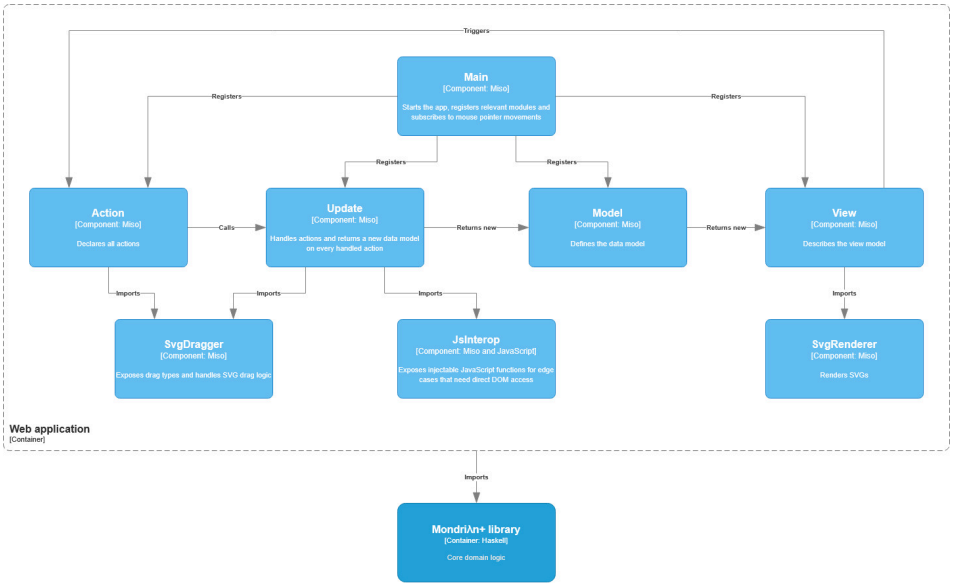


Figure 3.3: C4 app component view diagram

Lib Component View

Figure 3.4 shows a component level view of the Miso independent domain logic code, implemented as a standalone library exposing the depicted modules.

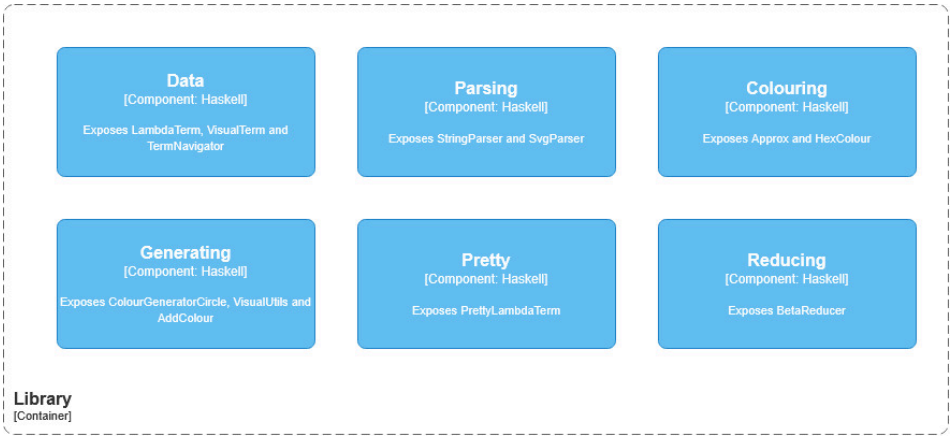


Figure 3.4: C4 lib component view diagram

Term Structure

This section highlights the central data structures used by the application.

`AnnotedTerm` is the core AST structure, that our code operates on. Its structure is equivalent to a regular lambda term but allows for an annotation value to be stored at every node. `VisualTerm` is simply a type alias for `AnnotatedTerm VisualInfo`. Within the `VisualInfos`, all the information regarding how a term is presented within an image is stored.

Information on whether to parenthesize terms or not and whether to render them in the curried or uncurried style is also stored in the `VisualInfos`. An alternative approach for this functionality would have been to modify the underlying `LambdaTerm` type in such a way that it:

1. stores input variables of abstractions as a list instead of nesting the abstractions with only one variable per abstraction
2. stores parenthesized terms in a separate `parenthesized` node that contains a `LambdaTerm`

After discussing these two approaches with our advisor we realized that, even though both approaches work, they both are not the perfect solution to this problem. The perfect solution would be an approach that allows for a seamless transition/mapping between the pure lambda term and the intermediate term containing additional information.

Unfortunately we did not have enough time to investigate this matter further and therefore, continued with the `VisualInfos` approach.

A working prototype of the alternative approach is implemented on the ast-way branch.

Comments

E.5 Web Framework Evaluation

On this page

In the evaluation phase of the project, an in-depth analysis of both Miso [6] and Reflex-DOM [9] was conducted. Both frameworks were recommended by our advisor as promising candidates for developing reactive web-apps in Haskell.

Miso

Miso is a compact and lightweight front-end framework for Haskell, that is heavily inspired by Elm [7] and React, designed to build reactive web applications. It provides a type-safe approach to state management and allows for unidirectional data flow. Miso also offers support for SVGs, enabling the creation of dynamic and interactive graphics within the virtual DOM.

Reflex-DOM

Reflex-DOM is a Haskell framework for functional reactive programming that also supports the development of interactive web applications. It is based on the Reflex-FRP engine. It puts strong emphasis on the paradigm of Functional-Reactive-Programming [8]. Usually the devlopment of Reflex-DOM applications is done via the Obelisk Framework, which is built on top of Reflex. Obelisk allows to quickly setup a project-skeleton and manages the needed dependencies. Manually setting up a Reflex-DOM project can be quite troublesome.

Conclusion

A small sampler app was set up for each framework to gain hands-on experience and enable comparison. After evaluation, Miso was selected because it is more lightweight, has a better documentation and seems to provide better SVG support than Reflex-DOM. In addition, Reflex-DOM seems to be more complex to set up and has a steeper learning curve.

Further evaluation revealed that Reflex-DOM shares the same shortcomings as Miso: it lacks certain built-in browser integrations and requires explicit JavaScript interop (for example, to implement file downloads).

Comments

E.6 Testing

On this page

For automated testing the HSpec framework is used, it supports HUnit-style unit tests, property-based tests, and automatic test discovery. A basic suite of automated tests (unit and property-based) is run as part of development/CI and complemented with manual system tests to verify overall functionality and subsystem interaction from an end-user perspective before merging to `master`.

Automated Tests

The architecture of Mondriλn+ allows for good testibility. All the domain logic code can be tested completely independent of the framework since it is capsuled into an importable library. The Miso application component code contains little to no domain logic and is therefore not worth testing with automated tests. The CI/CD pipeline contains a job that runs the tests on every commit.

All test code is located under `/test` and the files follow the `...Spec.hs` naming convention. Test discovery is enabled by the `Spec.hs` file, which only contains the entry point for the automatic test discovery.

Coverage report

The pipeline job that runs the tests runs them with the `--enable-coverage` flag in order to automatically generate a test coverage report. This test coverage report then gets deployed at the end of the pipeline to GitLab pages.

System Tests

Before merging into the `master` branch, the code on the different `development branches` is subjected to a manual system test to verify overall functionality from an end-user perspective. Testers manually exercise the implemented functional requirements to answer the question: Does the website behave as a user would expect?

Beyond UI and integration checks, system tests validate core Mondriλn+ functionality:

- Are β-reductions, term parsing and renaming performed correctly?
- Do the SVG's correctly represent their corresponding term?
- Does exporting/importing correctly preserve the state of the Terms?

If all checks pass, the system test is successful and the branch may be merged

Comments

E.7 CI/CD Pipeline

The CI/CD pipeline is executed by the GitLab Runner on the OST GitLab instance and performs automated formatting checks, tests, builds and automatically deploys artifacts.

Stages & Jobs

The pipeline, as depicted in Figure Figure 6.1, consists of the following stages and jobs.

format stage

Runs on every commit. Executes jobs related to code formatting.

format stage jobs

- fourmolu-check: Runs the fourmolu format checker on all Haskell source code. Fails the pipeline if the formatting check fails.

test stage

Runs on every commit. Executes jobs related to testing.

test stage jobs

- lib-test: Builds the library, runs the HSpec test suite and generates a coverage report.

build stage

Runs on every commit. Executes jobs related to building any kind of deliverables and artifacts.

build stage jobs

- wasm-app: Builds the WebAssembly bundle of the application using Nix (nix develop + make). Produces the static site artifacts.
- project-documentation: Builds the project documentation PDF from the LaTeX source using latexmk.

deploy stage

Only runs on commits to the main branch. Executes jobs related to publishing any kind of deliverables and artifacts.

deploy stage jobs

- pages: Collects artifacts (documentation PDFs, application bundle, test coverage report) and publishes them to GitLab Pages.

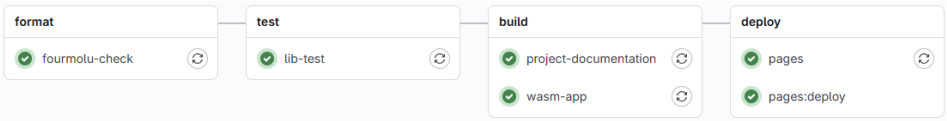


Figure 6.1: *Screenshot of the CI/CD pipeline*

Tooling

The pipeline uses the following tools to execute its jobs.

- fourmolu: Code formatting checks.
- Cabal and GHC: Build Code and run HSpec tests, produce coverage reports.
- Nix and make: Build the WebAssembly bundle (reproducible dev/build environment).
- latexmk: Compiles the project documentation PDF.
- GitLab Pages: Host static site, documentation PDFs and coverage report.

Published artifacts

The pipeline publishes the following artifacts (via the `pages` job).

- Project documentation PDF
- Web application bundle
- Test coverage report

Known Issues

Although the pipeline jobs are computationally not that complex, the executions still take quite long (generally 8 to 10 minutes for the entire pipeline) because Nix builds almost all dependencies from source every time.

This behavior is a common limitation and has been observed in other Haskell repositories using Nix. An attempt was made to use Cachix (a binary cache for Nix packages) to reduce Nix build times. However, the usage of Cachix did not produce any measurable improvement. We also tried to cache the dependencies directly on GitLab in such a way that the next pipeline execution can fetch the prebuild dependencies from the previous execution. But that also did not produce measurable improvements. Despite further troubleshooting efforts, the issue could not be resolved.

Comments

E.8 Bibliography

[1] Friedrich, Anja and Panchaud, Mona (2024). MondriΛn: A Visual Programming Language Based on the Lambda Calculus. OST Ostschweizer Fachhochschule.

[2] Wikipedia contributors (2025). Esoteric programming language. Wikipedia. https://en.wikipedia.org/wiki/Esoteric_programming_language (visited on 30th September 2025)

[3] Cornell University CS 312 contributors (2008). CS 312 Recitation 26: The Lambda Calculus. Cornell University. https://www.cs.cornell.edu/courses/cs312/2008sp/recitations/rec26.html (visited on 1st October 2025)

[4] Prof. Dr. Mehta, Farhad (2024). Functional Programming: Lambda Calculus Basics. OST - Ostschweizer Fachhochschule.

[5] Grady, Robert B. and Caswell, Deborah L. (1987). Software metrics: Establishing a company-wide program. Prentice-Hall

[6] dmjio (2025). Miso: A tasty Haskell web and mobile framework. https://github.com/dmjio/miso

[7] Evan Czaplicki (2025). elm: A delightful language for reliable web applications. https://elm-lang.org/

[8] Wikipedia contributors (2025). Functional reactive programming. Wikipedia. https://en.wikipedia.org/wiki/Functional_reactive_programming (visited on 14th of October 2025)

[9] Obsidian Systems (2025). Reflex DOM. https://reflex-frp.org/

[10] Simon Brown (2025). The C4 model. Visualizing software architecture. https://c4model.com/

Comments

E.9 Table of figures

Comments