Query-Driven Language Server Architecture using Second-Order Abstract Syntax

Danila Danko, MS-SE [1] Nikita Strygin, MS-SE [1]

Supervisor: Nikolai Kudasov [1]

[1] Innopolis University

November 13, 2024

Context

- The Language Server Protocol [1] standardized the protocol of communication between source code editors and language servers that provide language features like auto complete.

Context

- The Language Server Protocol [1] standardized the protocol of communication between source code editors and language servers that provide language features like auto complete.
- There are language servers for mainstream languages such as C#, Java, C++ [2]. Such servers improve the developer experience by supporting the "go to definition", "lookup the type on hover", "rename all occurences" queries.

Problem

- Multiple programming languages are implemented in Haskell [3], [4], [5].

Problem

- Multiple programming languages are implemented in Haskell [3], [4], [5].
- Only a half of the GitHub repositories for programming languages implemented in Haskell provide a language server [6].

Problem

- Multiple programming languages are implemented in Haskell [3], [4], [5].
- Only a half of the GitHub repositories for programming languages implemented in Haskell provide a language server [6].
- There exist frameworks for TypeScript (`Langium` [7]) and Python (`lsp-tree-sitter` [8]) that simplify integration with the LSP for new languages.

Problem

- Multiple programming languages are implemented in Haskell [3], [4], [5].
- Only a half of the GitHub repositories for programming languages implemented in Haskell provide a language server [6].
- There exist frameworks for TypeScript (`Langium` [7]) and Python (`lsp-tree-sitter` [8]) that simplify integration with the LSP for new languages.
- However, to our best knowledge, there is no such framework for languages implemented in Haskell!

Solution

- Language servers share some features, such as "go to definition".

Solution

- Language servers share some features, such as "go to definition".
- We assume that some of these features can be implemented in a Haskell framework that is independent of the target language (language-agnostic) for which LSP integration is developed.

Solution

- Language servers share some features, such as "go to definition".
- We assume that some of these features can be implemented in a Haskell framework that is independent of the target language (language-agnostic) for which LSP integration is developed.
- A promising approach is to work with the Second-Order Abstract Syntax (SOAS) [9] of the target language.

Solution

- Language servers share some features, such as "go to definition".
- We assume that some of these features can be implemented in a Haskell framework that is independent of the target language (language-agnostic) for which LSP integration is developed.
- A promising approach is to work with the Second-Order Abstract Syntax (SOAS) [9] of the target language.
- The main idea of SOAS is to provide a language-agnostic machinery for describing variable introduction in AST.

Solution

- Language servers share some features, such as "go to definition".
- We assume that some of these features can be implemented in a Haskell framework that is independent of the target language (language-agnostic) for which LSP integration is developed.
- A promising approach is to work with the Second-Order Abstract Syntax (SOAS) [9] of the target language.
- The main idea of SOAS is to provide a language-agnostic machinery for describing variable introduction in AST.
- This allows it to provide generic mechanisms for scope resolution ("go to definition"), variable bindings ("lookup the type on hover"), and substitution ("rename all occurences").

Contribution

- We want to implement in Haskell a language-agnostic framework that simplifies integration with LSP of (new) languages written in Haskell.

Contribution

- We want to implement in Haskell a language-agnostic framework that simplifies integration with LSP of (new) languages written in Haskell.
- The framework will primarily be based on the `free-foil` [10] (SOAS manipulation), `BNFC` [11] (parsing), `lsp` [12] (library for building LSP) packages.

Metrics

- We plan to provide integration with LSP for Simply typed lambda calculus (STLC) [13] and Stella Core [14] with (Approach 1) and without (Approach 2) our framework.

Metrics

- We plan to provide integration with LSP for Simply typed lambda calculus (STLC) [13] and Stella Core [14] with (Approach 1) and without (Approach 2) our framework.
- We will measure for both approaches and compare:

Metrics

- We plan to provide integration with LSP for Simply typed lambda calculus (STLC) [13] and Stella Core [14] with (Approach 1) and without (Approach 2) our framework.
- We will measure for both approaches and compare:
  - The lines of code required for integration and some other complexity metrics.

Metrics

- We plan to provide integration with LSP for Simply typed lambda calculus (STLC) [13] and Stella Core [14] with (Approach 1) and without (Approach 2) our framework.
- We will measure for both approaches and compare:
    - The lines of code required for integration and some other complexity metrics.
    - The performance of the server on a large (probably generated) code base (approx. 10KLoC).

Metrics

- We plan to provide integration with LSP for Simply typed lambda calculus (STLC) [13] and Stella Core [14] with (Approach 1) and without (Approach 2) our framework.
- We will measure for both approaches and compare:
  - The lines of code required for integration and some other complexity metrics.
  - The performance of the server on a large (probably generated) code base (approx. 10KLoC).
- Condition for success: metrics for the Approach 1 are at most 10% better than metrics for the Approach 2.

1. Introduction
2. Contribution
3. Evaluation criteria
4. Plan of work
   - Review the literature
   - Preliminary results
   - Practice with Free Foil
   - Practice with Simply typed lambda calculus (STLC)
   - Practice with Stella Core
   - Evaluate the results
5. References

Review the literature

- Read papers on SOAS [9], Foil [15] and Free foil [16].
- Read the `free-foil` package documentation [10].

Preliminary results

- We created a repository [17] for the thesis work.
- We implemented a parser and pretty-printer of STLC as defined in [18] using BNFC.

Practice with Free Foil

- Complete exercises on AST manipulation provided by Nikolai.
- Complete these exercises using Free foil.

Type checker and interpreter for STLC

- Implement a type checker and interpreter for STLC.
- Features:
  - Single module on the input and the output.
  - Use BNFC for parsing and pretty-printing.
  - Use Free Foil and Template Haskell.
  - Show error location in the code.

Language server for STLC

- Implement a language server and a simple VS Code extension for STLC.
- Use the `lsp` [12] package.
- Features:
  - Go to definition.
  - Type on hover.
  - Maybe something else that makes sense for STLC.

Type checker and interpreter for Stella Core

- Implement a type checker and interpreter for Stella Core.
- Features:
    - Single module on the input and the output.
    - Use BNFC for parsing and pretty-printing.
    - Use Free Foil and Template Haskell.
    - Show error location in the code.

Language server for Stella Core

- Implement a language server, and a simple VS Code extension for Stella Core.
- Use the `lsp` [12] package.
- Features:
  - Type and documentation on hover.
  - Autocompletion.
  - Inlay (type) hints.
  - (Un)folding scopes.
  - InfoView - a separate panel that shows context (what is in scope, the type of the current expression or goal)
  - Something else.

Evaluation

- Generate large code bases (approx. 10KLoC) for STLC and Stella.
- Evaluate the results using the Evaluation criteria.

[1] “Language server protocol.”
Available at https://en.wikipedia.org/w/index.php?title=Language_Server_Protocol&oldid=1236113985.

[2] “Lsp implementations.”
Available at https://microsoft.github.io/language-server-protocol/implementors/servers/.

[3] “Search results on github.com for "programming language language:haskell".”
Available at https://github.com/search?q=programming+language+language%3Ahaskell&type=repositories.

[4] “Hackage packages in the "formal languages" category.”
Available at
https://hackage.haskell.org/packages/#cat:Formal%20Languages.

[5] “Hackage packages in the "language" category.”
Available at https://hackage..haskell.org/packages/#cat:Language.

[6] "Search results on github.com for "language.lsp.server language:haskell"."
Available at `https://github.com/search?q=Language.LSP.Server+language%3Ahaskell&type=code`.

[7] "Langium."
Available at `https://langium.org`.

[8] "neomutt/lsp-tree-sitter."
Available at `https://github.com/neomutt/lsp-tree-sitter`.

[9] M. Fiore and D. Szamozvancev, "Formal metatheory of second-order abstract syntax."
Available at `http://arxiv.org/abs/2201.03504`.

[10] "free-foil package on hackage."
Available at `https://hackage.haskell.org/package/free-foil`.

[11] "Bnfc package on hackage."
Available at `https://hackage.haskell.org/package/BNFC`.

[12] "lsp package on hackage."
Available at `https://hackage.haskell.org/package/lsp`.

[13] "Simply typed lambda calculus."
Available at `https://en.wikipedia.org/w/index.php?title=Simply_typed_lambda_calculus&oldid=1252118159`.

[14] "fizruk/stella: Structurally typed extensible language for learning ACCPA."
Available at `https://github.com/fizruk/stella`.

[15] D. Maclaurin, A. Radul, and A. Paszke, "The foil: Capture-avoiding substitution with no sharp edges," in *Proceedings of the 34th Symposium on Implementation and Application of Functional Languages*, pp. 1–10, ACM.
Available at `https://dl.acm.org/doi/10.1145/3587216.3587224`.

[16] N. Kudasov, R. Shakirova, E. Shalagin, and K. Tyulebaeva, "Free foil: Generating efficient and scope-safe abstract syntax." Available at `https://arxiv.org/pdf/2405.16384`.

[17] “A repository for the ms thesis query-driven language server architecture using second-order abstract syntax.”
Available at `https://github.com/deemp/query-driven-free-foil/`.

[18] J. Dunfield and N. Krishnaswami, “Bidirectional typing.”
Available at `http://arxiv.org/abs/1908.05839`.

Thank You