UDK 004.4'232

Implementing a Language Server for the Rzk Proof Assistant

Abounegm A. (Innopolis University)

Kudasov N.D (Innopolis University)

The Riehl-Shulman type theory for synthetic ∞-categories is a new theory building on Homotopy Type Theory (HoTT). The experimental proof assistant Rzk offers an automated proof checker for this theory. With the goal of making this theory more accessible to mathematicians and computer scientists, we present in this paper a work-in-progress on a collection of command line and interactive tools for Rzk. These tools comprise a language server, an accompanying Visual Studio Code (VS Code) extension, and a list of smaller satellite tools offering minor conveniences. Although we focus on the support of VS Code, the language server is also compatible with other popular editors that support the Language Server Protocol (LSP), such as Emacs and Vim.

Keywords: Language Server Protocol, Visual Studio Code extension, proof assistant, homotopy type theory, category theory

1. Introduction

Proof assistants, also known as interactive theorem provers (ITPs), are software tools used in mathematics, computer science, and formal methods to assist in the development and verification of mathematical proofs. These tools play a crucial role in ensuring the correctness and reliability of complex mathematical statements and software systems. Examples of such tools include Agda [1], Coq [2], and Lean [3].

Although similar to typechecking in programming languages, proof checking is normally seen as an interactive process between the mathematician and a proof assistant. To this end, most proof assistants provide some interactive features either through specialized IDEs (e.g. CoqIDE[1]), integrating via Proof General [4], or Visual Studio Code (via the Language Server Protocol (LSP) [5]). Most well-established proof assistants support several of these options.

Rzk [6] is a new proof assistant that is based on Riehl-Shulman's type theory for synthetic ∞-categories [7]. The proof assistant is experimental but has been successfully used recently to formalize some fundamental results for ∞-categories, including the ∞-categorical Yoneda

[1] `https://coq.inria.fr/refman/practical-tools/coqide.html`

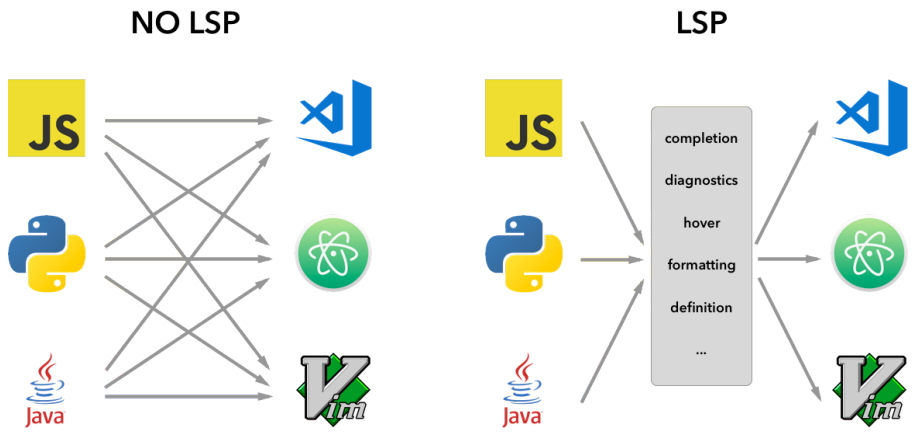

Fig. 1. The motivation behind LSP, from Microsoft's website. [2]

lemma [8].

1.1. Language Servers

A few years ago, when a new code editor was introduced, it needed to support the most popular programming languages, and depended on plugins written specifically for this editor to support languages not built into the editor. This led to a lot of repetitive work to provide useful language feature for the same language on multiple editors, and an inconsistency between the provided language features on different editors. The idea of language servers is Microsoft's attempt to solve this problem by standardizing a protocol for communication between a code editor (the client) and a background process (the server) that provides the language features for a certain language. This protocol is known as Language Server Protocol (LSP) [5]. With LSP, a language author need only implement the server once and would automatically get language support on any editor that supports LSP. Likewise, an editor that supports LSP would automatically get language support for any programming language that has an LSP server. Examples for the language features in question include providing diagnostic messages, jumping to the location an identifier is introduced, text completion, semantic syntax highlighting, and much more.

This protocol is particularly useful for interactive theorem provers since they rely on the editing experience much more than a command line interface. This is due to the fact that theorem provers generally do not need to compile to any kind of executable file and only need the type-checking stage of compilers, which can be easily performed by a language server. However, this also adds extra work on the language server since it needs to support more

[2] `https://code.visualstudio.com/api/language-extensions/language-server-extension-guide`

features than a regular compiler would, including listing variables in context along with their types, supporting Unicode symbols, and most importantly allowing for an interactive step-by-step proof walkthrough.

1.2. Related Work

Specification Language Server Protocol. While LSP has greatly improved the experience of developing language servers, it still leaves a bit to be desired. LSP was mainly designed with general purpose programming languages in mind, but theorem provers (or more generally, specification language) have slightly different requirements that are unmet by LSP. This is why [9] attempts to extend the original LSP specification with features that are especially useful for specification languages. The authors call the protocol extension *Specification Language Server Protocol* (SLSP). Simply speaking, it defines a set of new LSP requests/notification along with their payloads, and extends VS Code's interface with a view that displays proof information in a way similar to Proof General [4].

VS Code extension for Lean 4. Lean 4 [3] is a programming language and proof assistant by Microsoft Research[3] from which we draw much inspiration. In particular, the primary way to develop Lean programs is using its VS Code extension that provides a user interface for working with Lean interactively. This extension uses LSP to communicate with the Lean server and provides a lot of useful features, the most notable of which is the Info View panel that displays information about current proof state and allows interacting with it [10].

Proof General. Proof General [4] is a tool for developing interactive theorem provers that has been used for many widely-known proof assistants such as Coq and Isabelle. It is based on the Emacs editor and provides an interactive GUI with relative ease for theorem provers developed with its help.

1.3. Contribution

In this paper, we report on the work-in-progress on the implementation of utility and interactive tools around Rzk proof assistant, focusing on the language server and VS Code extension support.

[3] `https://www.microsoft.com/en-us/research/project/lean/`

2. Rzk Language Server and VS Code extesion

In this section, we describe design and implementation of the language server and the VS Code extension for Rzk proof assistant. The language server has direct access to the proof assistant internals, including the typechecking algorithm and the internal abstract syntax representation, and provides an interface conforming to the Language Server Protocol. The VS Code extension then acts as a buffer between the editor (VS Code) and the language server, to bring the interactive capabilities to the user.

We subject the language server to support the features specified below.

Intuitive Interface and Syntax Highlighting. The VS Code extension introduces an intuitive interface that aligns with the expectations of mathematicians and computer scientists. Users benefit from clear and accessible navigation, enabling efficient exploration of HoTT-based structures. Furthermore, the extension provides syntax and semantic highlighting, enhancing code readability, and facilitating error detection.

Code Completion and Suggestions. Rzk's VS Code extension leverages the LSP to offer intelligent code completion and context-aware suggestions. As users work with Rzk, the extension assists in writing code more efficiently by providing relevant suggestions, reducing the likelihood of syntax errors, and accelerating the development process.

Real-time Error Checking. One of the extension's notable strengths lies in its ability to perform real-time error checking. As users input and modify code, the language server continuously analyzes it, reporting back any type errors or other issues with the proof. This proactive error checking mechanism empowers users to identify and rectify issues promptly, fostering the creation of mathematically sound programs.

2.1. VS Code extension

To bring about these features to the users, a thin wrapper around the language server in the form of a VS Code extension is necessary. Additionally, the extension manages the installation of the language server itself on all major operating systems (Windows, macOS, and Ubuntu) powered by pre-built binaries attached to releases on GitHub, in addition to facilitating building the language server from source on platforms for which pre-built binaries are not available. It is also worth mentioning that the language server is designed to be compatible with any other

editor that supports LSP. In particular, users have reported success in integrating it with the NeoVim editor, and it is planned to be tested with other editors as well.

2.2. Rzk Language Server

At the core of the Rzk tool suite is the language server that powers all the editor features that make it pleasant to develop proofs in Rzk. In particular, it currently supports semantic highlighting, diagnostic messages, and text completion. Additionally, progress reporting for long-running processes (such as type-checking for large projects) is currently under work.

For the first versions of the language server, it is shipped as part of the Rzk proof assistant itself under a different subcommand, but it is planned to decouple both components and have the language server depend on the core library. They are implemented in the Haskell programming language using the `lsp`[4] package.

3. Conclusion and Future Work

We have designed and implemented an initial prototype of the language server and VS Code extension for an experimental proof assistant Rzk. An intermediate result of this work has been presented at the *Interactions of Proof Assistants and Mathematics*[5] school in Germany in September of 2023. This has helped gather feedback from the users and react to it on the spot. We feel that current users are mostly satisfied with the prototype tooling, but have also provided useful suggestions for further improvements.

In the future, we plan to support displaying variable information on hover, jumping to definition, renaming symbols, and formatting Rzk code. Eventually, we also plan to support rendering topes as images, and adding an information WebView similar to the one provided by Lean 4 [10].

It is also anticipated that a Haskell library for developing language servers (especially for proof assistants) can grow out of this project.

References

1. Bove Ana, Dybjer Peter, Norell Ulf. A brief overview of Agda–a functional language with dependent types // Theorem Proving in Higher Order Logics: 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings 22 / Springer. — 2009. — P. 73–78. — Access mode: `https://doi.org/10.1007/978-3-642-03359-9_6`.

[4] `https://hackage.haskell.org/package/lsp`

[5] `https://itp-school-2023.github.io/`

2. Bertot Yves, Castéran Pierre. Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions. — Springer Science & Business Media, 2013. — Access mode: https://doi.org/10.1093/comjnl/bxh141.
3. Moura Leonardo de, Ullrich Sebastian. The Lean 4 Theorem Prover and Programming Language // Automated Deduction – CADE 28 / ed. by Platzer André, Sutcliffe Geoff. — Cham : Springer International Publishing. — 2021. — P. 625–635.
4. Aspinall David. Proof General: A Generic Tool for Proof Development // Tools and Algorithms for the Construction and Analysis of Systems. — Springer Berlin Heidelberg, 2000. — P. 38–43.
5. Gunasinghe Nadeeshaan, Marcus Nipuna. Language Server Protocol and Implementation. — Apress, 2022.
6. Kudasov Nikolai. Rzk: a proof assistant for synthetic ∞-categories. — 2003. — Access mode: https://github.com/rzk-lang/rzk.
7. Riehl Emily, Shulman Michael. A type theory for synthetic ∞-categories // Higher Structures. — 2017.
8. Kudasov Nikolai, Riehl Emily, Weinberger Jonathan. Formalizing the ∞-categorical Yoneda lemma. — 2023. — 2309.08340.
9. The Specification Language Server Protocol: A Proposal for Standardised LSP Extensions / Rask Jonas Kjaer, Madsen Frederik Palludan, Battle Nick, Macedo Hugo Daniel, and Larsen Peter Gorm // Electronic Proceedings in Theoretical Computer Science. — 2021. — Aug. — Vol. 338. — P. 3–18.
10. Nawrocki Wojciech, Ayers Edward W., Ebner Gabriel. An Extensible User Interface for Lean 4 // 14th International Conference on Interactive Theorem Proving (ITP 2023). — 2023. — Vol. 268. — P. 24:1–24:20.