THE UNIVERSITY OF HONG KONG

Programming Language with Novel Type Inferences

Agrim Somani, Dave Mathew, Karanveer Singh, Shaheer Ziya

DETAILED PROJECT PLAN

Project Background

Type inference is a crucial aspect of programming language design, enabling the automatic deduction of data types in a program.

However, there exists a significant research gap between the theoretical design of type inference algorithms and their realistic implementations within programming languages, especially with regards to higher rank polymorphism.

Recent research has made significant strides in progressing the theory for higher rank polymorphic type inference, such as bidirectional type checking algorithms for higher-rank polymorphism [1] and work list-based higher rank polymorphic type inference [2-4].

Although the barebones implementations of the algorithms do exist, they often lack essential features, such as: a REPL, parser, tools to integrate the algorithm within existing languages (such as loading and type-checking files), meaningful error reporting or printing the typing derivations from type inferences.

This project aims to provide robust implementations for the aforementioned type inference algorithms and integrate them within a programming language. The project will also include a REPL, parser, and error reporting tools to provide a complete programming environment for developers to experiment with the algorithms.

These algorithms are essential for enhancing the expressiveness and reliability of programming languages, allowing developers to write more concise and less error-prone code. Furthermore, implementing these algorithms will help validate the practicality and efficiency of the algorithms in real-world scenarios, hopefully leading to more research in the field.

Objectives

A set of core objectives will be established for this project, committed to delivery. These core objectives will form the backbone of our programming language implementation. Additionally, a set of extra objectives will include supplementary features planned to be added on top of the foundational implementation, provided time and progress permits.

Before starting the implementations, a thorough literature review will be conducted to understand the theoretical underpinnings of the algorithms and their practical implementations. This involves understanding the specific notations used within such literatures and identifying the essential features that need to be implemented in the project.

Core Objectives

The main deliverable of our project will be an implementation of the state-of-the-art type inference algorithms specified in the literatures, built using Scala in the form of a programming language.

The implementation will come with a parser, REPL and a pretty printer. Programmers will be able to load scripts from other files to be executed, and obviously the implementation will be able to type-check the files based on the type inference algorithms.

The REPL will also be built in such a way which prints the typing derivations derived from the inference algorithms. Alongside type checking, the team will integrate a robust error reporting system within the programming language.

Extra Objectives

To enhance the applicability of the programming language, the project will explore new language features to incorporate. This includes adding the simple notion of modules, or extending the types supported by our language, to include algebraic datatypes and records.

Furthermore, the type inference algorithms itself could be made more robust. One possible way could be reengineering the work-list structures used within the algorithms to improve the scoping of the approach or to try uncover a monadic structure.

The extra objectives are bound to change as the project progresses through out the year.

Methodology

The project will start by formalising the Simply Typed Lambda Calculus (STLC) as a minimalistic language that will be used to implement the type inference algorithms. The STLC will be based on the languages formalised in the papers and will include the core concepts of the type inference algorithms.

A natural choice for the implementation of the STLC would be Haskell, as it is a functional programming language that is well-suited for implementing type inference algorithms with support for pattern matching, but Scala may be more appropriate considering the fact that it allows for side-effects that can be useful for optimising the algorithms. Additionally, Scala also has has good support for Monads (and other functional programming constructs) that leaves the doors open for us to re-engineer (or refactor) work-lists to possibly uncover a monadic structure.

The parser for the STLC is planned to be implemented using a parser combinator library, such as *fastparse* or *Parboiled2* in Scala, to allow for easy extensibility and maintainability. The parser will be used to parse the input programs and convert them into an abstract syntax tree (AST) that can be used by the type inference algorithm. Moreover, such libraries provide a high-level of abstraction that allows for easy composition of parsers and error handling.

The REPL could then be implemented using the parser and the defined syntax for the STLC. The REPL will allow users to interactively enter programs and see the type inference results in real-time. This will help in testing the type inference algorithm and providing immediate feedback to the user.

If time allows, the team would also work towards implementing the extra objectives such as an elaboration to a target calculus or the addition of other advanced features mentioned above. Although the details of their implementation are not yet clear, the team will discover and implement them as the project progresses.

However these are not essential for the core functionality of the project, they would provide additional value to the project and help in understanding the practical implications of the algorithms.

Project Schedule & Milestones

The project is to be completed incrementally, with the following milestones as our guide.

The project begins with a literature review of the papers and related work to understand the theoretical foundations of the type inference algorithms. The team would also be writing the implementations of the type inference algorithms while performing the literature review. This would involve understanding the algorithms and translating them into code. The team plans on completing this by the end of the first month.

The first major milestone would then be to model a complete type-checker for the STLC without type annotations, supported with a REPL, parser and pretty-printer for (algorithmic) derivations. From there, it will be important to identify the subset of rules that are relevant to the STLC and implement them in the chosen language. This will involve implementing the type inference algorithm, the REPL, parser, and pretty-printer for the STLC, as well as adapting the syntax from the papers accordingly. This would be mostly complete by the end of November.

Once the STLC and supporting tools have been implemented, our team will proceed to implement other features based that are a natural step forward from the STLC. This may include rethinking the implementations to possibly uncover some sort of a monadic structure, or have the algorithm return the inferred types, or even elaborate the STLC to a target calculus. These could be accomplished by the end of December or start of January.

BIBLIOGRAPHY

1. Jana Dunfield and Neelakantan R. Krishnaswami. 2013. Complete and easy bidirectional typechecking for higher-rank polymorphism. SIGPLAN Not. 48, 9 (September 2013), 429–442. https://doi.org/10.1145/2544174.2500582

2. Jinxu Zhao, Bruno C. d. S. Oliveira, and Tom Schrijvers. 2019. A mechanical formalization of higher-ranked polymorphic type inference. Proc. ACM Program. Lang. 3, ICFP, Article 112 (August 2019), 29 pages. https://doi.org/10.1145/3341716

3. Jinxu Zhao and Bruno C. d. S. 2022. Oliveira. Elementary Type Inference. In 36th European Conference on Object-Oriented Programming (ECOOP 2022). Leibniz International Proceedings in Informatics (LIPIcs), Volume 222, pp. 2:1-2:28, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.ECOOP.2022.2

4. Chen Cui, Shengyi Jiang, and Bruno C. d. S. Oliveira. 2023. Greedy Implicit Bounded Quantification. Proc. ACM Program. Lang. 7, OOPSLA2, Article 295 (October 2023), 29 pages. https://doi.org/10.1145/3622871