A Concurrent Runtime for the Pat Programming Language

Kai Wang - 2539930W

December 15, 2023

Status report

Proposal

Motivation

Concurrent programming is a critical and challenging area of modern software engineering, especially with the increasing demand for high performance and reliability. The supervisor mentioned that although Actor-based programming languages, such as Erlang and Elixir, offer significant advantages when dealing with concurrency, they still face problems such as communication mismatches and deadlocks in real-world applications. These challenges highlight the urgent need for a finer-grained, more reliable message-handling mechanism in concurrent programming.

Aims

The core goal of this project was to build a runtime system capable of handling the concurrent tasks of Pat, a new programming language based on the mailbox type, designed by the supervisor, who was the first to use mailboxes and who had achieved theoretical success through the development of a type checker. My approach to writing ”running” Pat programs focuses on crucial resource management functions such as process calls and rubbish collection. Implementing this runtime system will make the Pat language a complete toolset that can operate efficiently in natural software development environments and bring practical application value to concurrent programming.

Progress

- **Language and Tools Learning and Mastery:** Successfully learnt the syntax of the Pat language and its type-checking tool, mbcheck, and the fundamentals of OCaml and EIO, which provided the basis for a deeper understanding and implementation of the Pat language.
- **Theoretical studies and applications:** I read key literature on Mailbox Types, such as ”Special Delivery: Programming with Mailbox Types” and ”Mailbox Types for Unordered Interactions”, to deepen my understanding of concurrent programming.
- **Language understanding and experimentation:** Future variables in concurrent computation are explored through case studies and experimental code, especially in the context of message passing and state transitions.
- **Programming Practices:** Successfully applied the knowledge learnt from the CEK machine into OCaml code and attempted to apply this knowledge to the Pat language. Further, I developed functions to ”run” the Pat language and updated and maintained mbcheck on GitHub.

- **Problem Solving and Optimisation:** Solved problems encountered during the construction of mbcheck, such as the reduction rule for keywords in the Pat syntax, and explored the concept of Garbage Collection, particularly how to efficiently ”free” mailboxes and related issues.
- **Future goals:** It is planned to continue optimising the ”Free” feature and improve the type and structure of the mailboxes in mbcheck. In the meantime, the Pat language test cases will be thoroughly evaluated to ensure correctness and efficiency.

Problems and risks

Problems

- **Build and runtime environment issues:** Failure to build mbcheck was encountered, possibly due to complex dependencies or misconfiguration of the environment.
- **Functional language issues:** First-time user of OCaml and needed help understanding the syntax. Inefficiencies associated with the inability to use ”debug” mode to find the source of the problem.
- **Challenges in understanding Pat's syntactic structure:** Despite the learning curve, there were still difficulties in understanding Pat's syntax and reduction rules, especially when dealing with message types and mailbox operations.
- **Conversion of CEK machines to OCaml code:** There were challenges in applying concepts learnt from CEK machines to OCaml, particularly in adapting to the syntax and functionality of the Pat language.
- **Garbage Collection Optimisation:** Problems were encountered in implementing and optimising mailboxes' ”Free” functionality, particularly optimisation strategies for dealing with recursive procedures and function inter-calls.

Risks

- **Garbage Collection Optimisation Problem:** Multiple methods and concepts exist to learn and experiment with. Mitigation: Learn and apply in-depth step-by-step by focusing on a particular GC method's core concepts and functionality.

Plan

Semester 2

- Week 1-5: final GC optimization and evaluation tests run.
  - **Deliverable: the final generator source code, fully tested, ready for integration into the Pat language.**
- Week 5-10: Write up.
  - **Deliverable: first draft submitted to supervisor five weeks before final deadline.**