RIMPiler

6CCS3PRJ Background and Specification Progress Report

Author: Aidan Dakhama

Student ID: 21034945

Supervisor: Professor Maribel Fernandez

December 2023

Background and Context

0.1 Reversible Computing

Reversible computing is a model of computation in which the original state of the system can be recovered from the final state of the system[1]. Reversible computing is a wide-reaching, yet relatively under researched area of computer science with the potential to impact many areas of computer science, including low-power computing[2], high-speed computing, quantum computing, and more.

0.1.1 Classical Computing

In classical computing reversible computing has many applications, but one of the most prominent, which cannot be overcome by other means, is to reduce power consumption below the Landauer limit[2]. The ability to potentially reduce power consumption below the Landauer limit has many applications, including both low-power computing, which in the age of mobile computing is a very important area, and high-speed computing, which has applications in all performance critical areas of computing.

While we are still a ways from achieving this, as the landauer limit is around 100,000 time less energy per bit erasure than current computers. The study of all facets of reversible computing is key to achieving this goal.

0.1.2 Quantum Computing

Quantum computing allows us to solve problems which are intractable on classical computers, such as integer factorisation, which is the basis of many modern encryption algorithms. The quantum model of computing largely relies on the reversible model of computation in order to prevent decoherence.

Reversible computing allows the system to be run deterministically in either direction, ensuring there is no loss of information. In quantum computing, the conservation of information is vital to maintaining coherence of the system, and thus, reversible computation plays a key role in quantum computing.

0.2 Programing Languages

Programming languages are a key part of how we build systems, they abstract away much of the complexity or messiness of the underlying system, and provides us with a language which we can quickly build intuition and reason about. We can see in languages such as Janus[3] the language offers a high level inferable syntax which allows us to reason about a program. However, what it does not provide is a complete abstraction from the model of computation, this can be seen through restrictions in things like assignment. While this does not restrict the ability of the language, it does add additional barriers to developers, requiring them to adjust to the model of computation being used.

0.3 Compilers

Compilers are vital to working with high-level programming languages, applying translations to the code to allow it to be run on the target system which would be impractical to do manually for any non-trivial program. Additionally, compilers can do much more such as optimise the code, and provide feedback to the developer such as warnings and errors. This allows the developer to produce better code with less issues than it may have otherwise had.

Literature Review

What existing works are similar or tangential to this project? What are the relevant theories and concepts? What are the relevant technologies and techniques? What do we do differently?

Requirements

An overview of functional requirements, perhaps non-functional requirements, and a brief description of use cases.

Note, From a example submission: Ullah Joya Loyalty Card App 2015

”Importance of the requirements is shown next to them. A high priority requirement must be implemented for the project to be considered complete. Without low priority requirements, useful but not critical, the application should offer basic 23 functionality. High priority requirements should be implemented first and then tackle other lower priority requirements.”

So we should be able to specify even aspirational requirements, and then prioritise them.

Specification

Give requirements priorities, and specify them in more detail. perhaps also list limitations of the project, and assumptions made here.

Design

Identify use cases, with diagrams, give the system architecture, ”as low level as possible”. Use UML diagrams, data flow diagrams, sequence diagrams, etc.

Bibliography

[1] C. H. Bennett. “Logical Reversibility of Computation”. In: *IBM Journal of Research and Development* 17.6 (1973), pp. 525–532. DOI: `10.1147/rd.176.0525`.

[2] R. Landauer. “Irreversibility and Heat Generation in the Computing Process”. In: *IBM Journal of Research and Development* 5.3 (1961), pp. 183–191. DOI: `10.1147/rd.53.0183`.

[3] Christopher Lutz. “Janus: a time-reversible language”. 1986.