Project Proposal

Concepts of Program Design

Joris ten Tusscher, Joris Burgers, Ivo-Gabe de Wolff, Cas van der Rest, Orestis Melkonian

1 Background

- Reversible Computation
- Janus
- other languages
- a bit of theory

2 Problem

2.1 Experiment

2.2 Extensions

- Data structures (trees/sets/maps/ADTs)
- Prelude
- Syntactic sugar

2.3 Applications

- Pathfinding algorithms
- Encoding-Decoding
- Low-level bit manipulation (hamming error correction)
- [Optional] Reversible Debugger

2.4 Benchmarking

- Janus vs rFun
- LOC metrics

2.5 Formal verification

- Reversibility
- Verifying pre/post-conditions
- r-Turing Completeness

3 Methodology

3.1 Risk and contingency plans

There are a number of risks to consider when creating a DSL. These risks are enumerated below, with their respective contingency plans.

- **Arbitrary types** The plan is to implement arbitrary types into the DSL. The risk with this goal is mostly a time issue. It is expected that implementing arbitrary types will take a long time and therefore, the risk exists that the implementation can not be completed in the given timespan. The contingency plan, if this implementation takes to much time, is to reduce arbitrary types to some special types, losing the generality of the types.
- **Full verification** The goal is to completely verify the implementation of the DSL. To our knowledge, there is no implementation of a verified reversible language. Therefore, it would be an addition to provide a fully verified reversible language. If this full verification is not possible, the contingency plan is to verify a subset of the language instead of every feature.
- **Impossible applications** It is a possibility that the applications that are proposed to implement in the DSL are impossible to implement in a reversible language. If there is an application where there is no suitable variant found that can be implemented in a reversible language, that specific application will be ignored or a suitable replacement will be looked for.

- **Impossible use of libraries** There is the risk that one of the proposed libraries for implementing the DSL turns out not to be suitable. This may be the case because the library lacks some features that are necessary for the implementation of the DSL. If this is the case, a replacement will be looked for. If no suitable replacement can be found, the functionality will either be implemented in another way without the use of a library or will be removed from the specification.

- Template Haskell [1]
  - Embedding Janus
  - Compile-time guarantees
    - Type-checking
    - Variable usage

- GHC Profiling
  - Criterion package [2]

- Liquid Haskell [3]
  - Theorem proving

[1] https://wiki.haskell.org/Template_Haskell
[2] http://hackage.haskell.org/package/criterion)
[3] https://ucsd-progsys.github.io/liquidhaskell-blog/)

4 Planning

WEEK 1	**Task 1:** Orestis,...
	Task 2: Cas,...
WEEK 2	**Task 3:** Joris1, Joris2,...
	Task 4: Ivo,...
WEEK 3	**Task 1:** Orestis,...
	Task 2: Cas,...
PROGRESS REPORT	
WEEK 4	**Task 3:** Joris1, Joris2,...
	Task 4: Ivo,...
WEEK 5	**Task 1:** Orestis,...
	Task 2: Cas,...
WEEK 6	**Task 3:** Joris1, Joris2,...
	Task 4: Ivo,...
PROJECT SUBMISSION	