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 Janus is a programming language that is developed in 1982 by Chrisoper Lutz and Hower Derby [**?**]. The main feature of Janus is that the language is time-reversible. Time-reversible means that every statement in the language can be inverted. This means that at any moment, the program can be executed in reverse. This reverse execution can also be done if the program was not executed before the reverse execution. An example of this is the statement $x+ = 7$. This inverse of this statement is $x- = 7$. Even if the statement $x+ = 7$ was never executed, the program could be done backwards, given any valid final state of the program. Please note that not every operation can be inverted. Take for example the expression $x* = 0$. This expression has no inverse and is therefore not a valid Janus statement.
- other languages
- a bit of theory

2 Problem

2.1 Experiment

2.2 Janus

We want to implement the reversable programming language Janus as an DSL in Haskell. For the syntax and semantics the defenitions as given by Lutz en Derby [**?**] will be used. There will however be a few changes. There will be no input and output in the DSL. Therefore, the READ and WRITE

statements will not be valid statements. Beside this limitation, the DSL will also have the change that expressions can be any valid Haskell expression, in addition to the already valid Janus expressions. This includes function calls to Haskell expression. This gives a programmer the freedom to use existing Haskell code with the reversible DSL. This freedom of accepting Haskell code will not limit the reversibility of the DSL. This is because for a language to be reversible, the statement has to be reversible, not the expression. For example the statement $x+ = f$ where f is any valid Haskell expression. As long as f does not depend on x, the statement can be inverted by the statement $x- = f$.

2.3 Extensions

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

2.4 Applications

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

2.5 Benchmarking

- Janus vs rFun
- LOC metrics

2.6 Formal verification

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

3 Methodology

- Template Haskell [1]
  - Embedding Janus
  - Compile-time guarantees
    - Type-checking
    - Variable usage
- GHC Profiling
  - Criterion package [2]
- Liquid Haskell [3]
  - Theorem proving

4 Planning

- Milestones
- Division of labour

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