Rippl: Recursively Inferred Pure functional Programming Language

Da Hua Chen (Tester), Hollis Lehv (System Architect),
Amanda Liu (Language Guru), Hans Montero (Manager)

1 Motivation

Rippl is a functional language with the safety and elegance of pure languages like Haskell. With list comprehensions, a strong static type system implementing Hindley-Milner style inference, higher-order functions, and simple syntax, Rippl is a powerful computational language with strong support for list-oriented calculations as well as support for association of items of different types in the form of tuples. Rippl will be an appropriate introduction for users without prior functional programming experience who want to effectively and safely perform complex mathematical calculations.

This combination of features creates an incredibly powerful, expressive language for list-based data calculations and transformations with strong static guarantees for safety in a lightweight way that doesn't burden the user. The laziness also allows for the creation of massive lists and data executions that don't burden runtime or impede on safety unless they are evaluated, further giving the user more freedom in the way they write their programs.

2 Language Paradigms and Features

Rippl will be a declarative language with strong, static typing, static scoping, and lazy evaluation semantics.

Rippl will also feature higher order functions, and partial application. Rippl will also go beyond the Haskell-like suggested project by including the additional language features of type inference, immutability, and anonymous functions.

3 Hello World

This program demonstrates the use of type annotations and the `main` method entrypoint as well as the evaluation of certain expressions like mathematical integer addition.

```
1 main :: int
2 main = 1 + 1
```

4 Rippl in One Slide

Bertrand's Postulate stated in the Weak Prime Number Theorem that there is always a prime number to be found between some n and its double $2n$. This postulate was later proven by Pafnuty Chebyshev and refined by Paul Erdös. The following program includes a function that determines the primality of a number and a function that takes a n and returns the first prime between n and $2n$.

This program displays a lot of complex language features of Rippl including list comprehensions, lambda abstractions, type inference and annotations, and higher order functions.

```
1  prime_number_theorem :: int -> int
2  prime_number_theorem n =
3      let is_prime n =
4          let max = n / 2 in
5          let range = [2...max] in
6          let divisors = [x | x over range, n % x == 0] in
7          len divisors == 0
8      in
9      let range = [(n+1)...2*n] in
10     let odd_range = [x | x over range, x % 2 != 0 ] in
11     foldl
12         (fun prev -> fun curr ->
13             if is_prime prev then prev else curr)
14         (head odd_range)
15         odd_range
```