Browser-side Functional Programming

Second reader demo

Kieran Manning
09676121

April 3rd 2013

An Overview

The Problem?
- Javascript is unpleasant but ubiquitous. Javascript Suffers from..

- A lack of sane typing
- Verbose and inconsistent syntax
- A lack of laziness

These have been solved in languages such as Haskell, however it looks like we're stuck with Javascript for the forseeable future. What we need is a way of bringing some functional inspiration to browser-side programming via Javascript.

Implementation

An approach is needed that would...

- Bring lazy evaluation and type safety to Javascript
- Be modular, packageable and easily added.
- Be as efficient as possible
- Allow easy seperation of runtime evaluation from compilation
- Require no changes to Javascript itself

Core

Name given to a number of intermediate functional language representations, descended from System F lambda calculus, consisting notably of...

- Super combinator defintions and applications
- Variable names
- Integers
- Data constructors
- Case statements

GHC is capable of outputting a core-like language (see GHC external core) which is lazy and type-checked by GHC itself.

The G-Machine

The G-Machine, a functional compilation strategy.

- Historically significant but somewhat outdated.
- Takes a core-like language and produces a graph and sequence of instructions...
- ...designed to be evaluated by a minimal runtime.
- Designed to allow for easy adaptation of runtime in target languages.
- More efficient and modular than alternative Template Instantation approach, also explored.

A Graph Evaluation Runtime in Javascript

A runtime was required which would fulfill the following requirements:

- Take a representation of a G-Machine compiled graph state.
- Take a list of G-Machine evaluation instructions.
- Understand how to evaluate these instructions.
- Apply these instructions to the supplied graph and
- Return the resultant state of the graph.