Self-documenting computation

Olaf Klinke, Ph.D.

Lackmann Phymetric GmbH

Berlin, 28.2.2020

Section 1

Motivation

Me: Here is my invoice, please send the money.

They: Sure, but accounting needs to check it first.

Me: There are no mistakes, it is computer-generated.

They: Great, would you please hand over the spreadsheet file?

Me: Uh, there is no spreadsheet file. The calculation was done in `<fancy language>`. How else should I have verified the algorithm?

They: (long, baffled silence) What is `<fancy language>`? Never heard of it.
Look, our accountants need to know how you calculated the invoice. So please send step-by-step information.

Section 2

Problem statement

Scope

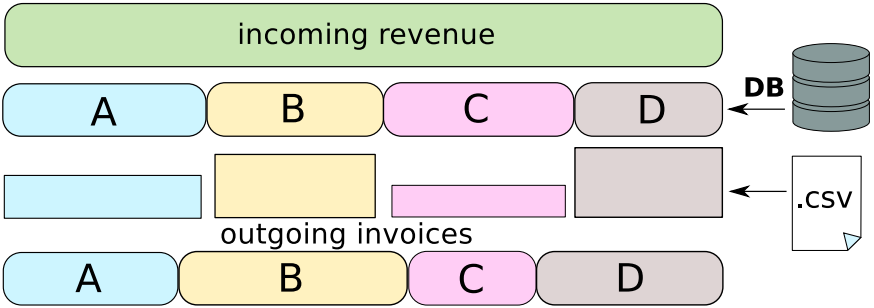
This presentation is for you if

- You perform non-trivial data transformations
- Your calculations must be verifiable by a human
- You can only make minimal assumptions about the verifying person
- Your data can be rendered in human-readable form

Toy example

Setup

Our company is a platform through which customers A, B, C and D sell goods. The monthly revenue for each company depends on the amount sold as well as some static weights.



Section 3

Demos

How to do it with Spreadsheets

(Demo)

Spreadsheets verdict

Pros

- Spreadsheet software is pervasive in the office world
- Interactive
- Intermediate values visible, data dependencies can be visualized
- I/O possible via ODBC or external file links

Cons

- You don't want to be debugging spreadsheet formulas
- Data model is limited compared to `<fancy language>`

How to do it with Haskintex

(Demo)

Haskintex verdict

Pros

- Literate programming, easy mixing of code and documentation

Cons

- Limited I/O functionality
- Cross-references must be inserted manually
- Might as well use HaTeX directly (documentation-within-code as opposed to code-within-documentation)

How to do it with IHaskell

(Demo)

IHaskell verdict

Pros

- Notebooks are intuitive to work with, batteries included™
- Interactive
- Powerful graphics capabilites
- Built-in export to HTML, PDF etc.

Cons

- Markdown cells can not reference data, manual workaround
- Only interactive when Jupyter and the backend is installed

Section 4

The Provenience package

The philosophy of provenience

- Documentation is only a small part of an actual application
- Computation and verification happen on different machines
- The actual algorithm must be extractable e.g. for testing
- Automatic tracking of data flow

How it works

Suppose you have an expression $g(f(x))$ where

$$x : A, \quad f : A \to B, \quad g : B \to C$$

Provenience registers x, $f(x)$ and $g(f(x))$ as nodes in a graph:

$$x \xrightarrow{f} f(x) \xrightarrow{g} g(f(x))$$

All nodes and labels are markup, either automatically generated via a type class or user-supplied.

The provenience monad transformer

The programmer turns a function

$$f : A \to M\,B$$

(where M is a monad of other side-effects, e.g. I/O) into a function

$$A \to (\mathit{StateT}\ \mathit{Graph}\ M)\ (B, \mathit{Node})$$

that is, a sub-computation alters the data flow graph and returns the result's node ID in the graph together with the actual data.

The provenience monad transformer building blocks

var

Register data as a node in the graph. Pairs of data and nodes are called *variables*.

render

Put a textual representation of the data at the node in the graph. The representation can be automatically generated or user-supplied.

describe

Provide a human-readable description of the data (optional).

apply

Apply a variable holding a function to a variable holding the argument, thus creating an edge in the graph.

How to do it with Provenience

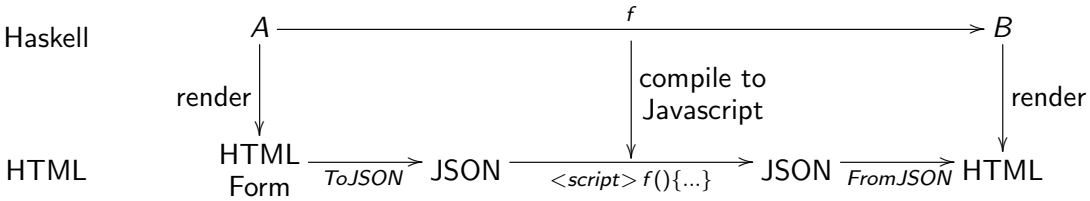
(Demo)

Section 5

Outlook

If I could...

I would make the output interactive (Maybe Fay can do this?)



1. Each input variable is rendered as a html `<form>` with given data as pre-filled content.
2. Clicking submit runs a Javascript version of the function f on the form data and updates the rendering of the output.
3. Re-computation of edges downstream of B are also triggered.