

Open Piping: a Visual Workflow Environment

Charles Boisvert

Sheffield Hallam University

c.boisvert@shu.ac.uk

1. Introduction

This submission presents our work on *Open Piping*¹, a visual workflow environment to make functional programming and data handling accessible to inexperienced learners.

2. Motivations

Four elements motivate our work: the growing ease of use and learning of programming tools; the rise of big data and data analytics, underpinned by functional programming; the visual model of functional computation; and finally the access barriers to this programming paradigm and to data science.

2.1. A systematic improvement in access to programming

Usability breakthroughs mark the progress of all computer science, including programming. One remarkable advance is the wide range of programming learning and novice developer environments, using a jigsaw puzzle metaphor to represent individual statements, such as MIT Scratch (Resnick et al., 2009).

2.2. The rise of data processing

While simple applications have become more accessible, computation has shifted to new domains, and to programming languages that support multiple paradigms, like R, Clojure, or Python which add functional programming to imperative, object-oriented and event based development. Yet, the jigsaw puzzle metaphor favours an imperative perspective on programming: the programming paradigms computing education tools support best, are becoming less used in professional practice.

2.3. Modelling functional computation visually

Lambda calculus' mapping to directed acyclic graphs provides a visual model, summarised table 1. The graph, or boxes-and-wires (hereafter BW) model, can read as a data flow.



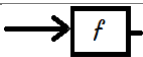
Notation	Represents	Graphical equivalent
x	Variable	
$\lambda x.f$	Abstraction (function f has parameter x)	
fx	Application (function f is applied to variable x)	

Table 1 – Basic elements of untyped λ -calculus and their representation as boxes and wires (BW)

2.4. Access limitations to visual functional programming

Data analysis applications require mastery of complex systems to apply mathematical techniques and represent information in non-trivial domains. Users', particularly novices, need carefully designed presentation and interaction devices. Below, we consider some variations that have been attempted.

3. Visual design

Coordinating code with result. The BW model represents computer code, but many visualisations offer multiple coordinated views of results, as Shneiderman and North (2000) propose. Yahoo pipes²

¹<http://boisvert.me.uk/openpiping>

²Discontinued by Yahoo, Inc. Wayback machine archive: <https://web.archive.org/web/20150604234337/http://pipes.yqlblog.net>

co-ordinates code with a sample of the resulting data. Selecting subsets of code supported debugging.

Data Typing. Viskell shows textual type annotations, and PROGRAPH visual shape clues, as seen in fig. 1 below. Yahoo pipes did not show types, but enforced type checking through interaction.

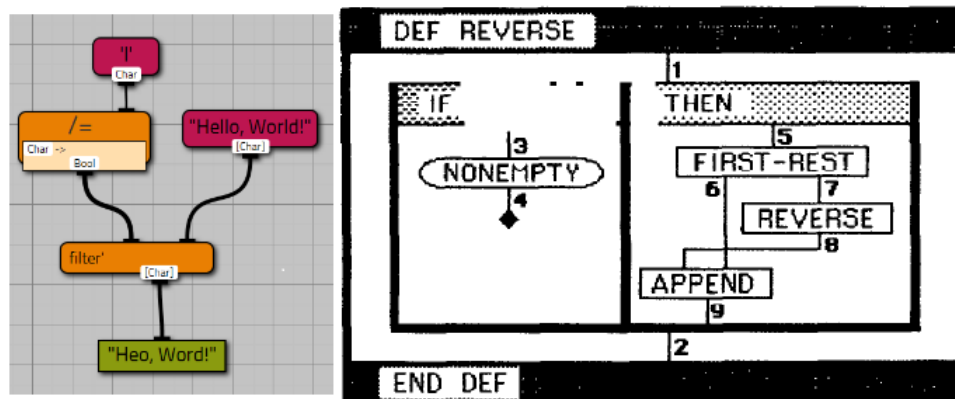


Figure 1 – BW in Viskell, left (Wibbelink, 2016) and PROGRAPH, right (Cox and Mulligan, 1985)

Representing Conditionals. A conditional execution function directly maps to a box and three wires, but this 'direct' solution may not be 'directly' understood by end-users. PROGRAPH uses a *frame* to divide code in chunks that end-users best consider separately.

First class functions. First-class functions' representation in the model is clear: a box represents a function, unless a wire shows its application to a variable. But as Viskell's use of this solution (pictured above) shows, a novice would not understand it without help. An alternative is to represent first-class functions within frames. Fukunaga *et al.*'s (1993) discusses a full representation of first-class functions with this solution. It shows the need for careful study to represent the notions in ways that users can understand and control.

4. Conclusion

We believe that Data Processing is inaccessible to the public, mainly due to a cognitive barrier, but that existing work shows ways to effect greater ease of learning, of use and availability to the functional languages and data analysis tools. To make the visual model support development by end-users, learning by novices, understanding by learners, we will need to propose and evaluate alternative solutions empirically.

References

- Cox, P. and Mulligan, I. (1985). Compiling the graphical functional language prograph. In *Proceedings of the 1985 ACM SIGSMALL symposium on Small systems*, pages 34–41. ACM.
- Fukunaga, A., Pree, W., and Kimura, T. D. (1993). Functions as objects in a data flow based visual language. In *Proceedings of the 1993 ACM Conference on Computer Science, CSC '93*, pages 215–220, New York, NY, USA. ACM.
- North, C. and Shneiderman, B. (2000). Snap-together visualization: A user interface for coordinating visualizations via relational schemata. In *Proceedings of the Working Conference on Advanced Visual Interfaces, AVI '00*, pages 128–135, New York, NY, USA. ACM.
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11):60–67.
- Wibbelink, F. (2016). Interacting with conditionals in viskell.