ComMotion: Using Animation to Illustrate Functional Programming Concepts

Barry Burd
Drew Univeristy
New Jersey, USA
bburd@drew.edu

By animating the evaluation of expressions in Haskell, we can help students embrace function evaluation as a programming paradigm. An animation framework that's written in Haskell can do double duty. It's both a learning tool and an example of a useful Haskell application.

1 Introduction

This is an informal, preliminary report on a project named **ComMotion** (short for **Com**position in **Motion**). The project uses animation to help students visualize the behavior of functional programming code. We've begun using ComMotion in a Haskell programming course for undergraduate Computer Science majors at Drew University.

The use of visualization to help students understand functional programming isn't new. In a formal study, Urquiza-Fuentes et al. [1] found measurable benefits from the use of animations to teach functional programming. More recently, Weck et al. [2] use data-flow diagrams to help programmers understand Haskell code.

The direct motivation for ComMotion comes from a long-standing tradition in algebra instruction – the tradition of visualizing the parts of an equation moving from place to place on a page. Neither of the previously cited papers (Urquiza-Fuentes et al. [1], Weck et al. [2]) apply animation to this kind of symbol manipulation.

Animations showing terms moving from one side of the equal sign to another appear on some tutorial websites. For example, the Animating Algebra page on the Juniverse website [3] has several animations showing the evaluation of $3x-2=2x+5$. These animations were created using Adobe Flash. Since Adobe Flash has no specific features for moving terms within algebraic expressions, this method for illustrating symbol manipulation doesn't scale very well.

2 First Steps

As a prototype, the author created visualizations using the animation features in Microsoft PowerPoint. The use of PowerPoint doesn't solve the scalability problem, but it helps us to see the kinds of animations that can be created. For example, to evaluate `filter even [4,3,6]`, a PowerPoint animation shows the `even` function gliding across the `[4,3,6]` list, eliminating the values for which `even` isn't true (see Figure 1).

We can visualize the `map` function in much the same way that we visualize the `filter` function in Figure 1. That is, we can slide a function along a list of values. But in some situations, it's more appropriate to show the `map` function taking an orthogonal slice out of a list of values (see Figure 2).

Submitted to:
TFPIE 2019

In Figure 1, when the `even` function moves off the rightmost edge of the list, it disappears, and the word `filter` disappears as well. Thus, the expression `[4,6]` replaces `filter even [4,3,6]`, which suggests that the original `filter even [4,3,6]` expression is being evaluated in place. This makes it easy to illustrate function composition by sequencing animations one after another. For example, in Figure 3, a simple animation illustrates the composition of `maximum` with `map`.

Informal feedback from the students in Drew University's **CSCI 335: Functional Programming** course (during the Spring 2019 semester) indicated that the PowerPoint animations were helpful. But with the semester already in progress, there was no way to rigorously test the animations' effectiveness.

Moving beyond the use of PowerPoint animation, the author created a sample animation using Haskell with Gloss [4]. The transition from PowerPoint has two purposes:

- **With Haskell code, we can script the animations.** This opens the door for building a full-fledged expression animation tool. We can use the tool to build custom visualizations of functional programming concepts.
- **After seeing the Haskell version of the animation, students can look "behind the curtain" to see how the visualization is created using the purely functional paradigm.** This is an eye-opener for students who use mostly text-based input and output in the introductory Functional Programming course.
- **Since Haskell can be used to create animations, there would be no excuse for ignoring it as the primary tool to animate Haskell programming techniques.** More generally, for all x and all y, the use of y for learning x is most compelling when $x = y$.

3 Future Work

The primary goal of this project is to build on the existing Haskell/Gloss code to create ComMotion – a full featured tool for animating expression evaluation in functional programs.

The next offering of Drew University's Functional Programming course is scheduled for the Spring 2020 semester. During that semester, the author will formally evaluate the effectiveness of ComMotion as a Haskell language learning tool.

An additional goal is to encourage students to create their own visualizations. One student imagines the motion in Figure 1 with a function sliding from element to element in a list. Another student imagines the motion in Figure 2 with values being distilled from a list of more exotic entries. ComMotion will help students understand the usefulness of such visualizations and will enable the implementation of their visualizations. Students will practice their Haskell programming skills by writing code in the ComMotion framework.

References

[1] J. Urquiza-Duentes & J. Velazquez-Iturbide (2012): *A Long-Term Evaluation of Educational Animations of Functional Programs*. Available at `https://ieeexplore.ieee.org/document/6268026`.

[2] T. Weck & M. Tichy (2016): *Visualizing Data-Flows in Functional Programs*. Available at `https://ieeexplore.ieee.org/abstract/document/7476651`.

[3] J. Lester (2002): *Animating Algebra*. Available at `http://thejuniverse.org/PUBLIC/MathDesign/AnimatingAlgebra/index.html`.

[4] B. Lippmeier (2018): *Gloss*. Available at `http://gloss.ouroborus.net/`.

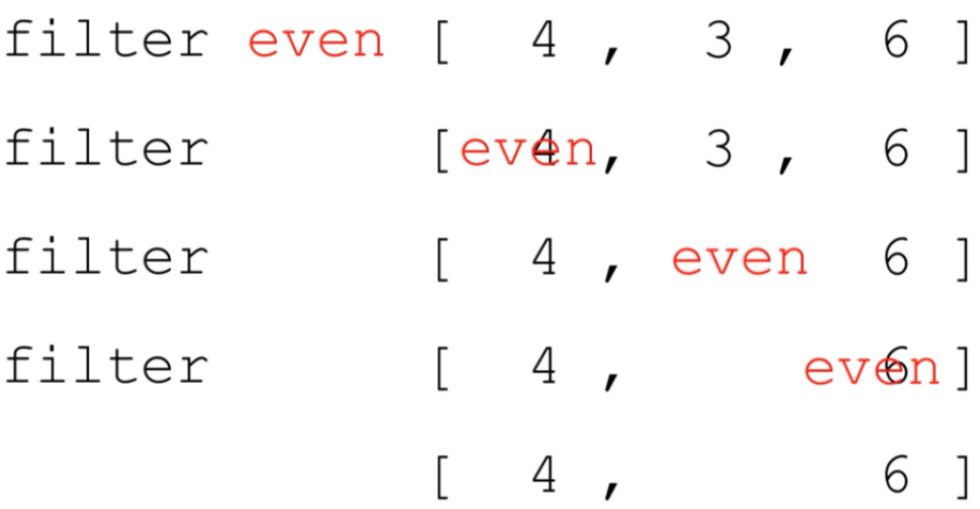

Figure 1: Applying `filter` to a list.

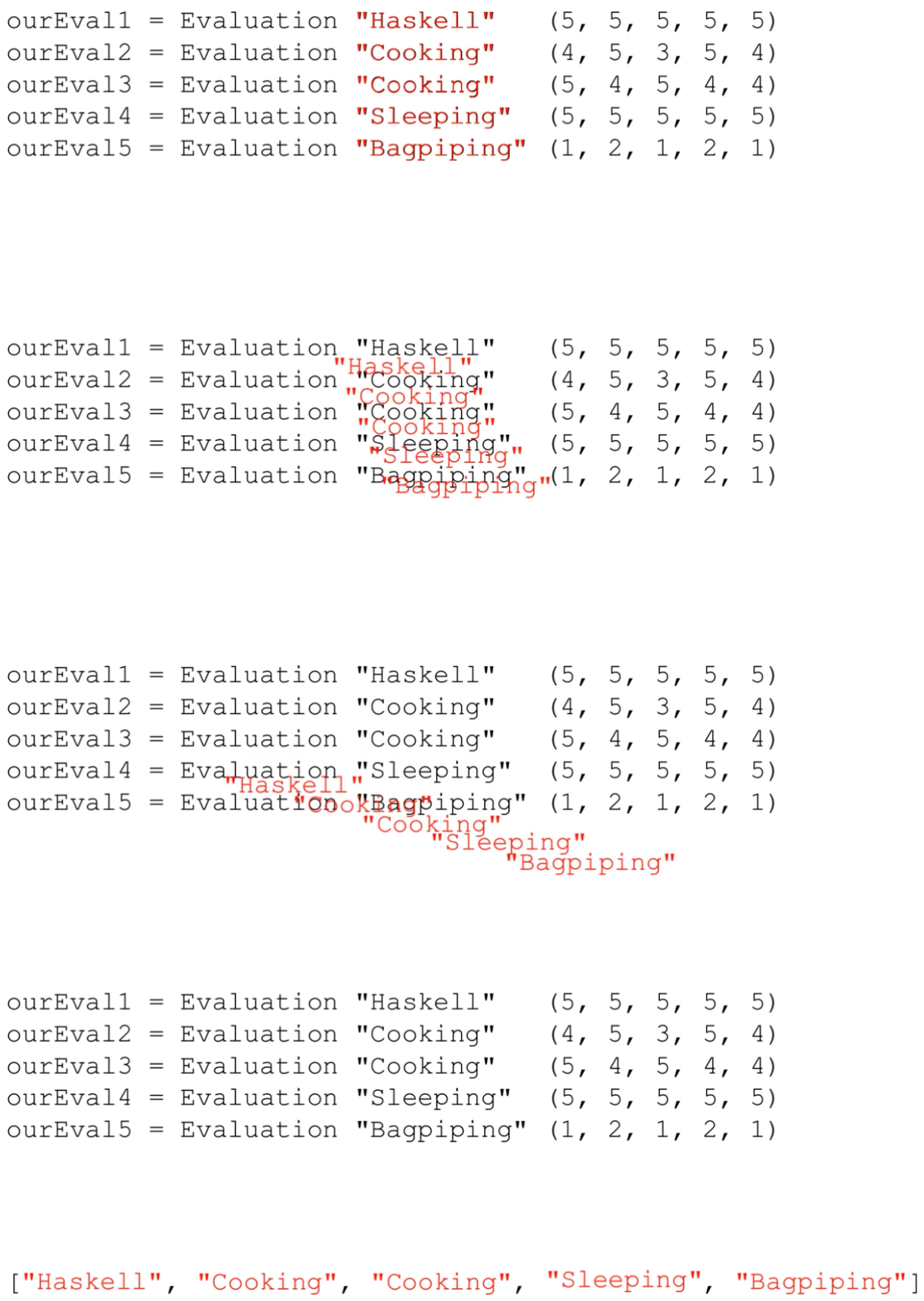

Figure 2: Getting a list of course names from a list of course evaluations using the `map` function.

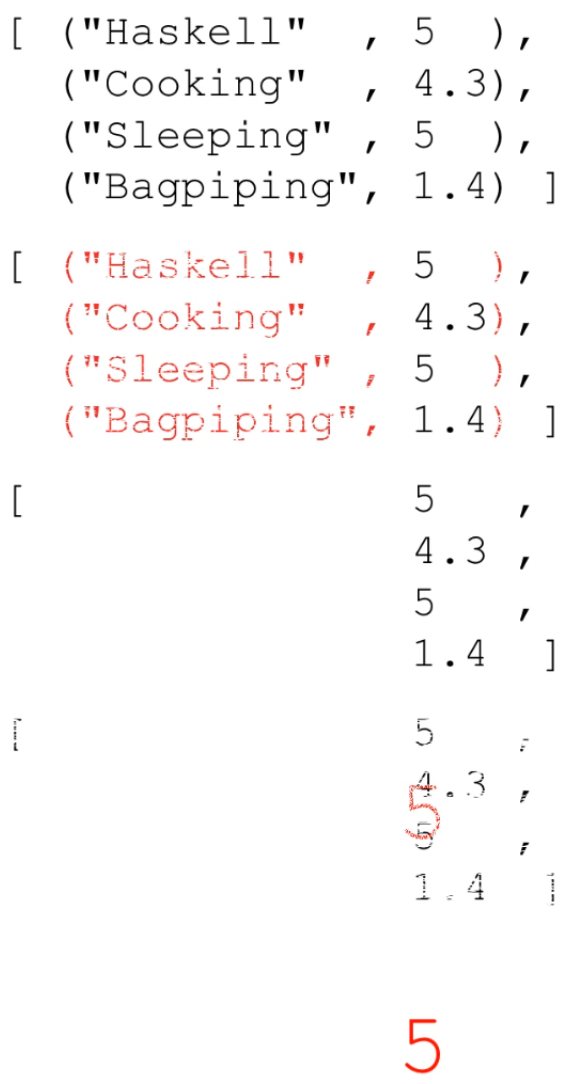

Figure 3: Finding the highest rating from a list of (course name, rating) tuples.