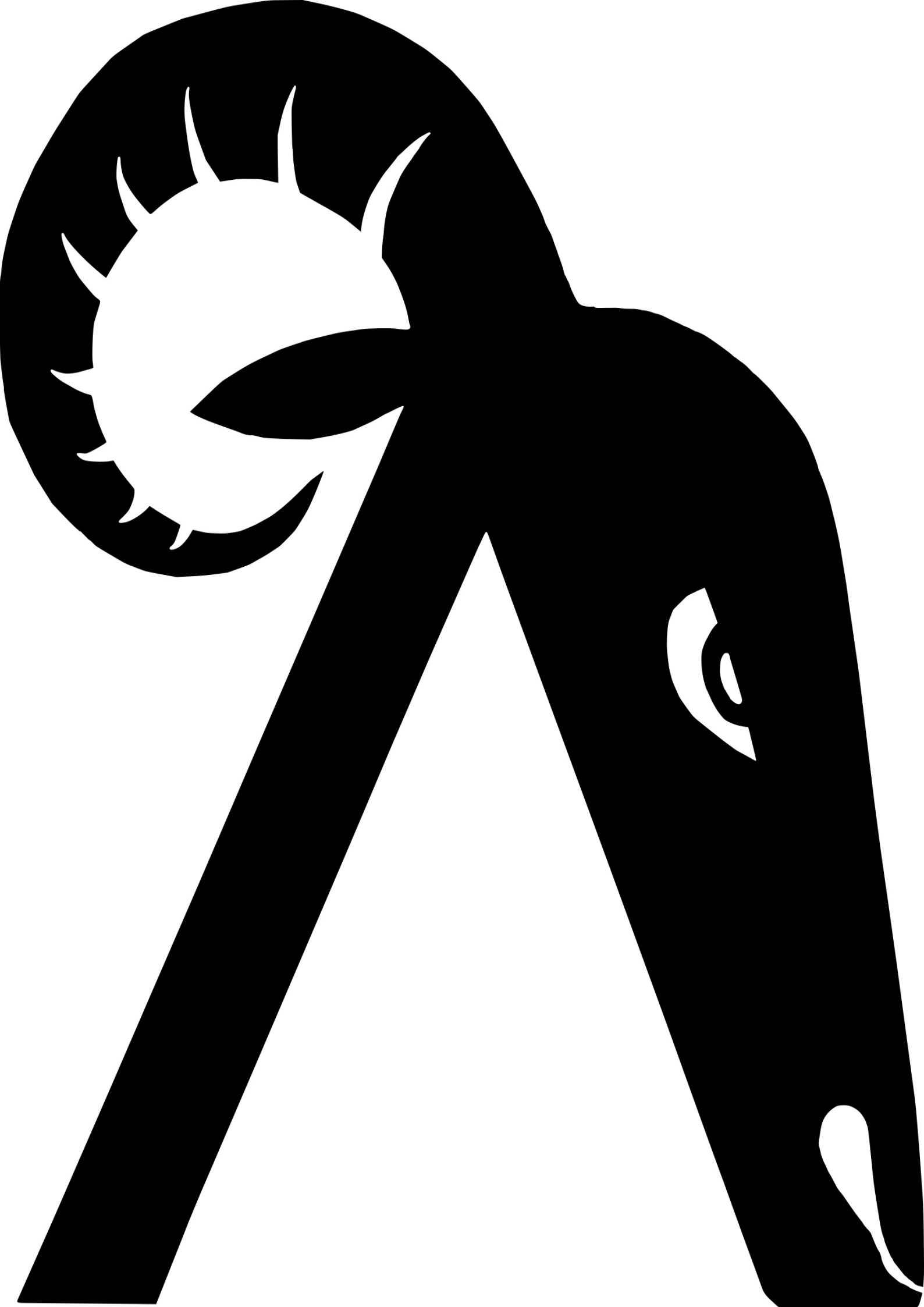

Table of Contents

1. Introduction
2. Introduction: Core features in Ramda
    i. Small functions against common data structures
    ii. Immutability of user data
    iii. Avoiding side-effects
    iv. Composing functions
    v. Currying everywhere
    vi. Dispatching
3. Working with functions
    i. Currying
    ii. Combining functions: `compose*`, `pipe*`, `converge`, `useWith`
    iii. Changing signatures: `flip`, `partial`, `lift`, `nAry`, `apply`, `unapply`
    iv. Calling functions: `apply`, `call`, `bind`, `memoize`
    v. Adding additional functionality
4. Working with lists
    i. Working with lists
    ii. Arrays as lists
    iii. Strings as (sometimes) close-enough
    iv. Building lists: `range`, `repeat`, `times`, `keys`, `toPairs`
    v. Slicing and dicing lists: `take`, `drop`, `append`, `prepend`, `concat`
    vi. Tweaking lists
    vii. List to list transformations: `map`, `pluck`, `filter`, `aperture`, `uniq`, `zip`, `partition`
    viii. Transforming lists to single values: `reduce`, `find`, `nth`, `all/any`

Ramda

Ramda is a "practical functional library for Javascript programmers."

It is designed to bring make of the standard techniques of functional programming to the world of Javascript. Its goals are modest: to make it easy to work with small, standard functions against consistent data types such as lists, to make it easy to work with immutable data, to make it easy to compose applications function-by-function, composing simpler functions into more sophisticated workflows.

Ramda also has a few more advanced techniques up its sleeve. Its delegation mechanism allows us to turn many OO-style APIs into more functional versions, its lenses allow us to simply focus attention on some specific part of a data structure, and its transducers allow us to combine many operations on every element of a list of data into a single operation per element.

But at its heart, Ramda is a simple library. It is not a framework that dictates how we build our applications. Instead it is a low-level toolkit that offers a collection of functional tools we believe to be helpful.

TODO

TODO

TODO

TODO

Composing functions

TODO

TODO

TODO

Currying

TODO

TODO

TODO

TODO

TODO

Working with lists

Arrays as lists

Strings as (sometimes) close-enough

Building lists: `range`, `repeat`, `times`, `keys`, `toPairs`

Slicing and dicing lists: `take`, `drop`, `append`, `prepend`, `concat`

Tweaking lists

List to list transformations: `map`, `pluck`, `filter`, `aperture`, `uniq`, `zip`, `partition`

Transforming lists to single values: `reduce`, `find`, `nth`, `all/any`