DECLARATIVE INTERACTION DESIGN FOR DATA VISUALIZATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Arvind Satyanarayan
July 2017

Abstract

This thesis tells you all you need to know about...

Acknowledgments

I would like to thank...

Contents

List of Tables

List of Figures

Chapter 1

Declarative Primitives for Interaction Design

Chapter 2

A Streaming Dataflow Architecture

Chapter 3

A Grammar of Interactive Graphics

3.1 Visual Encoding

The simplest Vega-Lite specification — referred to as a *unit* specification — describes a single Cartesian plot with the following four-tuple:

unit := (data, transforms, mark-type, encodings)

The *data* definition identifies a data source, a relational table consisting of records (rows) with named attributes (columns). This data table can be subject to a set of *transforms*, including filtering and adding derived fields via formulas. The *mark-type* specifies the geometric object used to visually encode the data records. Legal values include *bar*, *line*, *area*, *text*, *rule* for reference lines, and plotting symbols (*point* & *tick*). The *encodings* determine how data attributes map to the properties of visual marks. Formally, an encoding is a seven-tuple:

encoding := (channel, field, data-type, value, functions, scale, guide)

Available visual encoding *channels* include spatial position (x, y), *color*, *shape*, *size*, and *text*. An *order* channel controls sorting of stacked elements (e.g., for stacked bar charts and the layering order of line charts). A *path* order channel determines the sequence in which points of a line or area mark are connected to each other. A *detail* channel includes additional group-by fields in aggregate plots.

The *field* string denotes a data attribute to visualize, along with a given *data-type* (one of *nominal*, *ordinal*, *quantitative* or *temporal*). Alternatively, one can specify a constant literal *value* to serve as the data field. The data field can be further transformed using *functions* such as binning, aggregation (e.g., mean), and sorting.

An encoding may also specify properties of a *scale* that maps from the data domain to a visual range, and a *guide* (axis or legend) for visualizing the scale. If not specified, Vega-Lite will automatically populate default properties based on the *channel* and *data-type*. For x and y channels, either a linear scale (for quantitative data) or an ordinal scale (for ordinal and nominal data) is instantiated, along with an axis. For *color*, *size*, and *shape* channels, suitable palettes and legends are generated. For example, quantitative color encodings use a single-hue luminance ramp, while nominal color encodings use a categorical palette with varied hues. Our default assignments largely follow the model of prior systems [**?**, **?**].

Unit specifications are capable of expressing a variety of common, useful plots of both raw and aggregated data. Examples include bar charts, histograms, dot plots, scatter plots, line graphs, and area graphs. Our formal definitions are instantiated in a JSON (JavaScript Object Notation) syntax, as shown in

3.2 View Composition Algebra

Multiple *unit* specifications can be composed using the following operators:

- *layer([unit$_1$, unit$_2$, ...], resolve)* produces a view in which subsequent charts are plotted on top of each other. To produce coherent and comparable layers, we share scales (if their types match) and merge guides by default. For example, we compute the union of the data domains for the x or y channel, for which we then generate a single scale. However, Vega-Lite can not enforce that a unioned domain is *semantically* meaningful. To prohibit layering of composite views with incongruent internal structures, the *layer* operator restricts its operands to be *unit* views.

- *hconcat([view*$_1$*, view*$_2$*, ...], resolve)* and *vconcat([view*$_1$*, view*$_2$*, ...], resolve)* place views side-by-side horizontally or vertically, respectively. If aligned spatial channels have matching data fields (e.g., the y channels in an *hconcat* use the same field), a shared scale and axis are used. Axis composition facilitates comparison across views and optimizes the underlying implementation.

- *facet(data, field, view, channel, scale, axis, resolve)* produces a trellis plot [**?**] by subsetting the *data* by the distinct values of a *field*. The *view* specification provides a template for the sub-plots, and inherits the backing *data* for each partition from the operator. The *channel* indicates if sub-plots should be laid out vertically (*row*) or horizontally (*column*), and the *scale* and *axis* parameters enable further customization of sub-plot layout and labeling.

  To facilitate comparison, scales and guides for quantitative fields are shared by default. This ensures that each facet visualizes the same data domain. However, for ordinal scales we generate independent scales by default to avoid unnecessary inclusion of empty categories, akin to Polaris' *nest* operator. When faceting by fiscal quarter and visualizing per-month data in each cell, one likely wishes to see three months per quarter, not twelve months of which nine are empty.

- *repeat(values, view, channel, scale, axis, resolve)* generates one plot for each entry in a list of *values*. The *view* specification provides a template for the sub-plots, and inherits the full backing dataset. Encodings within the repeated *view* specification can refer to this provided *value* to parameterize the plot[1]. As with *facet*, the *channel* indicates if plots should divide by *row* or *column*, with further customization possible via the *scale* and *axis* components. By default, scales and axes are independent, but legends are shared when data fields coincide.

Each operator provides default strategies to *resolve* scales, axes, and legends across views, as described above. A user can choose to override these default behaviours by specifying tuples of the form *(channel, scale|axis|legend, union|independent)*.

[1] As the *repeat* operator requires parameterization of the inner view, it is not strictly algebraic. It is possible to achieve algebraic "purity" via explicit repeated concatenation or by reformulating the repeat operator (e.g., by including rewrite rules that apply to the inner view specification). However, we believe the current syntax to be more usable and concise than these alternatives.

These operators form an algebra: the output of one operator can serve as input to a subsequent operator. As a result, complex nested views and dashboards can be concisely specified. For instance, a layer of two unit views might be repeated, and then concatenated with a different unit view. The one exception is the *layer* operator, which, as previously noted, only accepts unit views to ensure consistent plots. For concision, two dimensional faceted or repeated layouts can be achieved by applying the operators to the *row* and *column* channels simultaneously. When faceting a composite view, only the dataset targeted by the operator is partitioned; any other datasets specified in sub-views are replicated.

Chapter 4

An Interactive Visualization Design Environment (VDE)

Bibliography