FUNCTIONAL PEARLS
Composing data visualizations

TOMAS PETRICEK
University of Kent, UK
(*e-mail:* `t.petricek@kent.ac.uk`)

1 Introduction

Let's say we want to create the two charts in Figure 1. The chart on the left is a bar chart, except that it shows two different values for each bar. The chart on the right is a line chart, except that it also highlights two parts of the timeline with two different colors.

There is a plenty of libraries that can draw bar charts and line charts, but adding those extra features will only be possible if the author already thought about your exact scenario. For example, Google Charts supports the left chart (it is called Dual-X Bar Chart) but there is no way for adding a background, let alone based on the X value. The alternative is to use a more low-level library such as D3. In D3 you construct the chart piece by piece, but then you have to tediously transform your values to coordinates in pixels. For scientific data visualizations, you could use an implementation of the Grammar of Graphics such as ggplot2. This lets you map data to geometric objects (such as points, bars, and lines) and their visual properties (X and Y coordinate, shape and color). However, the number of options is still somewhat limited.

To a functional programmer, this situation is disappointing. We should be able to compose a variety of charts from a small number of primitives. The shapes should be defined not in terms of pixels, but numbers of seats or exchange rates. We should also be able to easily derive high-level abstractions, such as a bar chart, from those low-level primitives.

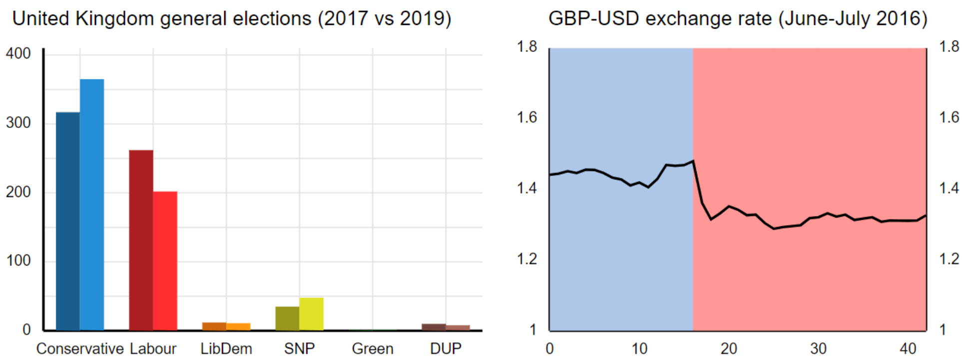

Fig. 1. Two charts about the UK politics: Comparison of election results from 2017 and 2019 (left) and GBP-USD exchange rate with highlighted areas before and after the 23 June 2016 Brexit vote.

In this paper, we...

2 Domain specific language

Domain-specific values

All values that are provided when constructing charts are from the chart domain. For example, when visualizing the population of different countries, X values will be categorical values such as United Kingdom or Czech Republic and Y values will be continuous values such as 66.04 million or 10.58 million. Compost will automatically map the value from a domain value to a pixel value. The user does not need to do any rescaling to obtain value in pixels. This is akin to most high-level charting libraries, but in contrast with some more flexible libraries like, say, D3. Compost supports only two-dimensional charts and values can be either categorical (such as different countries) or continuous (such as population size). A value needs to determine an exact position in the space available for the chart. For continuous values, this simply means applying a linear transformation. For categorical values, the situation is more difficult. For say we are drawing a bar chart with population of three different countries with countries on the X axis and population on the Y axis. The scale of the X axis is categorical and Compost divides the available space into three ranges of equal width. However, then United or Czech Republic does not determine an exact position but instead a range. To determine a unique position, we need to attach a value between 0 and 1 to the categorical value which identifies a relative position in the available range. This is illustrated in Figure 2. More formally, we write c for categorical

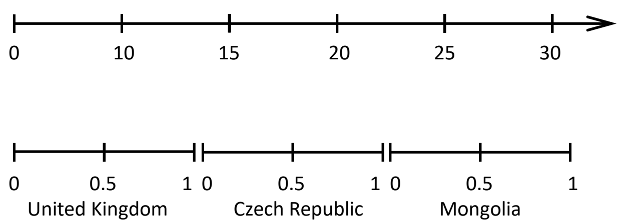

Fig. 2. Values on a continous scale and categorical scale.

values such as countries (represented as country names using strings), r for ratio between 0 and 1 and we write n for numerical continuous values with any range. A value v used in a Compost chart is defined as:

$$v \quad = \quad cat\ c, r cont\ n$$

A value v that is used to create Compost charts can be categorical, written as cat and consisting of the category and a ratio that determines point in the space allocated for the cateogry. A continous value, written as cont, consists of a number and will be mapped to a location based on the implicitly obtained re-scaling.

Language of composable charts

Now that we defined our representation of values, we can define a basic composable language of data visualizations. A shape s represents a chart. In the most basic form, it can be text label, line connecting a list of points and a filled polygon determined by a list of points. For line, text and polygon, we also include a parameter γ indicating the color to be used. In addition, a shape can be constructed by overlaying several other shapes:

$$\begin{array}{rcl} s & = & \mathit{overlay}\ s_1, \ldots, s_n \\ & | & \mathit{text}\ \gamma, v_x, v_y, t \\ & | & \mathit{line}\ \gamma, v_{x1}, v_{y1}, \ldots, v_{xn}, v_{yn} \\ & | & \mathit{fill}\ \gamma, v_{x1}, v_{y1}, \ldots, v_{xn}, v_{yn} \end{array}$$

As an example, consider bar chart with UK and CZ as two categorical values on the X axis and their population in millions (continuous value) on the Y axis. This can be constructed by creating two filled rectangles using fill and overlaying them using overlay:

$$\begin{array}{l} \mathit{overlay} \\ \quad \mathit{fill}\ \#0000ff,\ (\mathit{cat}\ UK, 0), (\mathit{cont}\ 00.00), (\mathit{cat}\ UK, 0), \qquad\qquad (\mathit{cat}\ UK, 1), (\mathit{cont}\ 66.04), (\mathit{cat}\ UK, \quad \mathit{fill}\ \# \end{array}$$

The shape specification overlays two bars of different colours. To construct a rectangle, we define four points. For the UK, two of the points have Y value set to 0 (bottom of the bar) and two have it set to 66.04 (top of the bar indiciating the population value). The values on the X axis are categorical UK values – two are on the left as indicated by the ratio 0 and two are on the right as indicated by ratio 1. The ratio is always relative to the category, so the ratios for the second bar are also 0 and 1.

Calculating scales and projections

When processing shape specification such as the one given in the previous section, the Compost library infers the scales for the X and Y axes and computes a projection function that can be used for mapping domain-specific values to pixels. While doing this, it also checks for consistency all values on one axis have to be either categorical or continuous, but not a mix of the two. The first step is to determine the scales for each of the axes. A scale can be continuous, defined by a minimal and maximal value, or a categorical, defined by a list of categorical values. Note that we do not need to track the ratios used as each category will be allocated equal amount of space, regardless of where in that space shapes need to appear. A scale l is thus defined as:

$$l \quad = \quad \mathit{categorical}\ c_1, \ldots, c_k \mathit{continuous}\ n_{min}, \ldots, n_{max}$$

A scale is obtained by recursively walking over the shape and gradually constructing two scales, for X and Y axis, from the X and Y coordinates that appear in the shape. The process is fairly straightforward and we won't discuss it in detail (for now). The first value encountered determines what kind of a scale will be constructed. If the type of a later value does not match the type of a scale, the process fails. Once we obtain scales for both X and Y axes, we calculate a projection function for each axis. The function takes a value v together with total space available (say, in pixels) and produces a position in pixels corresponding to v. We also omit the details here (for now), but briefly – for continuous values, we produce a simple linear transformation; for categorical values, we split the available space into k equally sized regions (where k is the number of categorical values in the scale) and then map a categorical value $\mathit{cat}\ c, r$ to the region corresponding to c according to the ratio r.

Controlling and nesting scales

The process of computing scales can be controlled by additional primitives in the Compost domain-specific language for describing shapes that were not discussed earlier. The following definition lists additional primitives that are also a part of the definition of s. Note that the primitives can be applied either to the X scale or to the Y scale - we write x/y to indicate that a sub-script on those primitives can be set either to x or to y (both scales can be modified by nesting two primitives):

$$\begin{array}{rcl} s & = & (\ldots) \\ & | & axis_{x/y} s \\ & | & roundScale_{x/y} s \\ & | & explicitScale_{x/y}\ l, s \\ & | & nest_{x/y}\ v_{min}, v_{max}, s \end{array}$$

We first focus on the first three primitives, while the nest primitive will be discussed in the next section. The axis and roundScale constructs could be defined as derived but it is easier to consider them as primitives for now. The axis primitive simply takes the s and draws an axis around the contents of s, using the inferred scale of s to determine displayed on the axis. This could be a derived construct, because axes can be rendered using lines and text labels. The roundScale operation takes the inferred X or Y scale of the shape s and, if it is a continuous scale, rounds its minimal and maximal values to "nice" numbers. For example, if a continuous scale has minimum 0 and maximum 66.04, the resulting scale would have maximum 70. For categorical scale, the operation does not have any effect. The explicitScale operation is similar, but it replaces the inferred scale with an explicitly provided scale (the type of the inferred scale has to match with the type of the explicitly given scale). For example, assuming populationBar is the example given earlier, we can write:

$$\begin{array}{l} axis_x\ (axis_y\ (roundScale_y \\ \quad (explicitScale_x\ (categorical \end{array}$$

This replaces the X axis with an explicitly given one that includes extra country code and also overrides the implicitly inferred order of the categorical values. We then ask Compost to automatically round the Y scale (showing population) so that we get a "nice" number as the maximum. Finally, we add axes around the shape, producing a usual labelled chart. As noted earlier, both axis and roundScale could be defined as derived – axis would have to infer the scales of the nested shape, calculate appropriate labels and insert suitable lines and labels; the roundScale operation would need to infer the scale too and then add an explicit explicitScale with a rounded minimal and maximal value of a continuous scale.

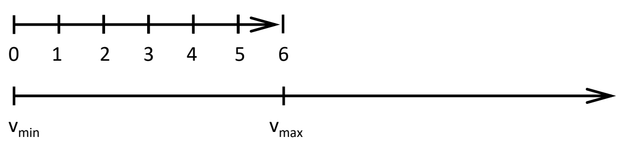

Fig. 3. A continuous scale with values from 0 to 6, nested in another scale.

Nesting of charts and scales

Finally, one more primitive that we added in the above definition and that we have not explained yet is $nest_{x/y}$. The parameters of the primitive are two values, v_{min} and v_{max} together with a shape s. The primitive makes it possible to nest one scale inside another one. When inferring the scales, Compost infers the scale of the nested shape s. This will then take the area determined by the two points v_{min} and v_{max} in the scale determined by other shapes that are overlaid with the *nest* shape. The two points are also used to determine the scales of the overall shape, but the scales of s are nested and do not have any effect on the outer scales. In fact, the scale of s does not even have to be of the same type as the scale of the outer shape. Figure 3 illustrates this with an example. Here, we are nesting a continuous scale with values ranging from 0 to 6 into another scale. The values v_{max} and v_{min} do not even have to be continuous. We can, for example, nest a line chart inside a bar of a bar chart and then the two values could be $cat\ One, 0$ and (which are two values that define a region of a categorical scale). One area where nesting of scales is particularly useful is when combining multiple charts in a single view. For example, let's say that we have two line charts showing prices of two different stocks over the same period of time (for simplicity, we can just use UNIX timestamp as a time). The prices of the two stocks are orders of magnitude different, so they cannot fit easily into the same chart. We want to show two line charts side-by-side (one above each other). They should each have their own Y axis (price), but the X axis (time) should be shared. Assuming we have googPrices and fbPrices as two line charts without any axes, covering the same date range, we can write:

$$
\begin{array}{l}
axis_x\ (overlay \\
\quad (nest_y\ (cat\ top-chart, 0), (cat\ top-chart, 1), (axis_y\ fbPrices)), \\
\quad (nest_y\ (cat\ bottom-chart, 0), (cat\ bottom-chart, (axis_y\ googPrices))\ )
\end{array}
$$

Reading the code from the inner-most part to the outer-most, we first add separate Y axes to both of the line charts. Given the difference in the prices, the axes will have quite different values. We then nest the (continuous) Y axes using the nest primitive and, at the same time, define a new outer Y axis. The outer Y axis is categorical with just two values, top-chart and bottom-chart. This means that the two nested charts will take equal amount of space, one above the other (each of the charts takes the full space allocated for the category – the minimal value has ratio 0 while the maximal value has ratio 1). [TODO: We really need illustrations for those example charts, but that would be more work!] [TODO: Maybe use this chart as a motivation: https://wellcomeopenresearch.org/articles/4-63/v1]

3 Random ideas

- We could define a type system to catch categorical/continuous value mismatch in a single
- It would be worth thinking about ordinal values, which are categorical but can be sorted.
- There might be other kinds of scales - for example, color scale (can have meaning in scatter or a scale for secondary markers (like sizes of bubbles in a bubble chart). We could really say every value (including bar chart colors) should be coming from some scale...

- Implementing something like axis or roundScale as an actual derived primitive is a bit tricky, because it needs to invoke a part of the normal rendering workflow (to run the automatic inference of scales on the nested shape) – this might be just implementation issue, but it could be some more basic problem.