Two Walk-through Cases: Bond Yields and Disaster Cost

Anonymous Author(s)

ABSTRACT

This document includes two in-depth walk-through cases of our examples: Disaster Cost (from mobile to desktop) and Aid Budget (from desktop to mobile) as part of the Supplementary Material of 'Cicero: A Declarative Grammar for Responsive Visualization.'

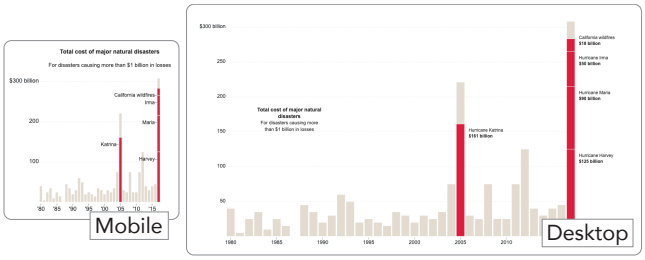
1 DISASTER COST (MOBILE TO DESKTOP)

The Disaster Cost chart depicts the losses associated with major natural disasters in the U.S. In Figure 1, the x position encodes the year, while bar height represents the loss in USD. Bars for five disasters are red and have labels (their names) in the mobile version. There is a title element (two lines) above the visualization. In addition to resizing, changes for the desktop view involve (1) internalizing the title, (2) using a longer form of disaster names ('Hurricane' is added) with the loss amount in USD, (3) adding y axis labels for 50, 150, and 250 billion, (4) lengthening the x axis labels (e.g., '80 to 1980). The Cicero spec is shown in Figure 1.

The resizing transformation in line 2–4 expresses `modify`ing the `size` of the `view` (reusing the previous rule but changing the size value). After resizing, there is a large empty space on the left side of the chart. One way to utilize that space is to internalize the existing title element into the chart, which can be done using the rule in line 5–10. In this case, one can use the `replace` action instead of `reposition` to express converting the title element to an internal non-data annotation. To indicate the new role, the `option` has the `to` keyword (line 8–9). The `internal` keyword in line 9 expresses that the converted annotation should appear inside of the visualization. However, this rule provides no designated position for the internalized annotation, so our Cicero compiler relies on the default behavior of the rendering grammar (our extended Vega-Lite) for choosing the center of the largest empty area (**P6**). The `separate` keyword set to `false` in line 10 means that the two title lines move together as a single annotation to prevent the automated positioning from separating them. This transformation can be done using the `reposition` action and `internal` keyword in the `option` as well. One may specify the absolute position of the internalized annotation using the `x` and `y` keywords.

The increased space can also add more axis labels and make labels longer to enhance the clarity of reference elements. The addition rule in line 11–14 `add`s the values of 50, 150, and 250 (in the `option` at line 13–14) to the vertical axis (`vAxis`), resulting in new axis labels and grid lines for the corresponding values (**P3**). The modification rule in line 6–10 lengthens the text format (`expression`) of the axis labels for the `year` field (the `specifier` in line 15–17). The `null` value for the `expression` keyword in line 19 means 'no particular format,' showing the original four digits for the years.

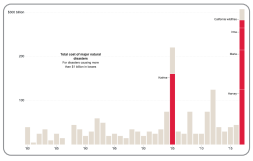
In addition to the larger screen space, desktop screens allow for utilizing the horizontal offsets of visualizations (e.g., the visualization margins). Therefore, the modification rule in line 20–28

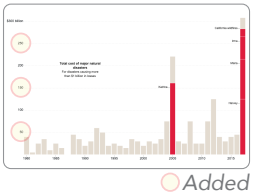


Cicero transformations

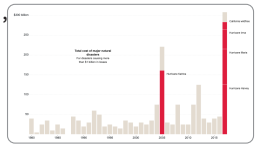
```
1  ...
2  { specifier: { role: "view" },
3    action: "modify",
4    option: { size: [1024, 612] }},

5  { specifier: { role: "title" },
6    action: "replace",
7    option: {
8     to: { role: "annotation",
9           internal: true },
10    separate: false }},

11 { specifier: { role: "vAxis" },
12   action: "add",
13   option: {
14     values: [50, 150, 250] }},

15 { specifier: {
16     role: "axis.label",
17     field: "year" },
18   action: "modify",
19   option: { expression: null }},

20 { specifier: { role: "mark.label" },
21   action: "modify",
22   option: {
23     dx: 10,
24     orient: "top-right",
25     anchor: "start",
26     align: "left",
27     tick: null,
28     expression: "(datum.value!=='California wildfires' ?
         'Hurricane '+datum.value : datum.value)" }},

29 { specifier: { role: "mark.label" },
30   action: "add",
31   option: {
32     items: [{
33       fontSize: 12,
34       fontWeight: 700,
35       expression:
           "('$'+datum.cost+' billion')" }]}}
36 ...
```

Transformed





Added



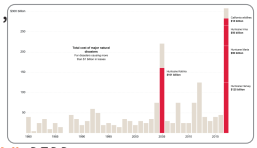


Figure 1: A walk-through example case of Disaster Cost (Section 1).

changes the position and text format of mark labels. In the `option`, line 23–25 modify the mark labels to be 10 pixels away from the corresponding marks horizontally (`dx`) using the `top-right` position of each mark and the `start` of each label as reference points (`orient` and `anchor`, respectively). Line 26 sets the text `align`ment as `left` as the labels are moved to the right of the marks. As it becomes easier to distinguish the relationship of each mark-label pair

https://nyti.ms/38VBIzz

due to proximity and the added spacing between labels, line 27 indicates that no label ticks are needed for the desktop view (as a shortcut for {specifier: "mark.label.tick", action: "remove"}). To lengthen the mark labels, line 28 changes their expression using a JavaScript-based expression meaning "if the label value is not 'California wildfires' then add 'Hurricane' before it with a space."

In addition to lengthening the mark labels, one can add more information like the loss cost of each of the specified disasters to the existing mark labels, as expressed in line 29–37. In line 32, the items keyword in option indicates subelements of the element selected by the specifier (i.e., text lines of each mark label). The add action in line 30 adds the specified items as part of the mark labels. Line 33–34 sets text style properties; other properties like text alignment does not need to be specified as the compiler looks for those properties from other similar elements (i.e., the existing labels; **P3**). Lastly, line 35 formats the new text element as "$00 billion" using the expression keyword.

2 AID BUDGET (DESKTOP TO MOBILE)

The Aid Budget example summarizes the COVID-19 aid budget plans for each business sector suggested by the Democratic (blue) and Republican (pink) U.S. parties compared to the budgets already passed by the U.S. Congress (lightgray). The desktop version in Figure 2 has three columns for the Republican, already passed, and Democratic plans in this order, and rows for eight business sectors, composing a tabulated format. Each bar encodes the budget amount of the corresponding plan and sector categories. To address the significantly reduced screen width in mobile screens, transformations for mobile include (1) changing the chart layout to a single-column grouped bar chart with two row fields (sector → plan), (2) converting the axis for plan to a legend for the color channel, (3) reordering the plan row, and (4) miscellaneous changes to text elements. The Cicero spec is shown in Figure 2.

To fit the mobile screen, the resizing rule in line 2–4 modifies the size to 375 × 350. This transformation results in an overly narrow resolution for the horizontal bars. To address the narrowed column width, one can partially transpose the column field (plan) to the row, as indicated in line 5–9. In the option, the from keyword (line 8) specifies the column at index of 0 (the second column is the budget value field). The to keyword (line 9) indicates the row element at index of 1, which does not exist before the transformation (i.e., after the first row element). Using this index keyword instead of the field keyword enhances the reusability of this partial transpose rule by making it agnostic to the underlying data set. This rule is equivalent to using a specifier of {role:"column", "index": 0} instead of having the from keyword.

By changing the layout as above, one might want to change the order of bars to emphasize the "already passed" data as a reference point for both plans (previously shown at the center of the desktop view). In the mobile view, one can emphasize those reference points by placing them above the other bars. To enable such a layout transformation, the reordering rule in line 10–14 first specifies the the row of the plan field (line 10). This specifier reflects the previous change from the row to the column (**P7**). Then the sort

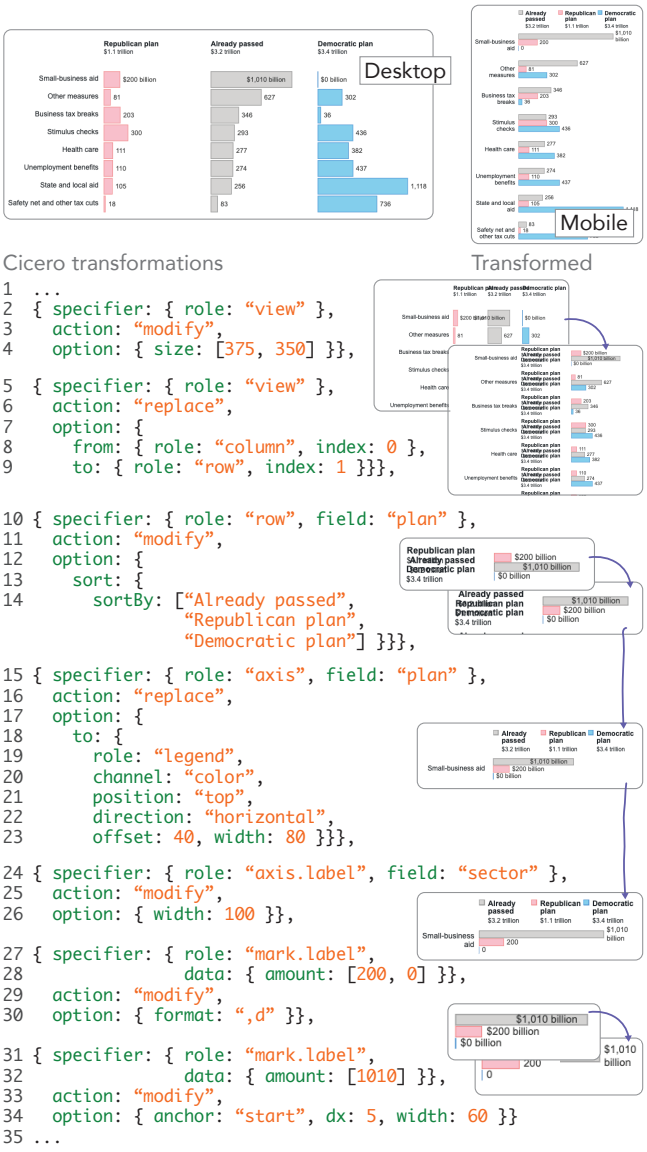


Figure 2: A walk-through example case of Aid Budget (Section 2).

keyword and its subproperty sortBy in the option indicate the new order of the field (line 13–14).

The previous changes cause the vertical axis labels for the plan field to overlap with each other and repeat for each sector (**P2**). At the same time, another encoding channel (color) also encodes the same field. Using this redundant encoding, one can change the axis labels to a color legend: the replacement rule in line 15–23 converts the axis for the plan field (the specifier in line 15) to the legend for the color channel (the option in line 19–20). The additional keywords (position, direction, offset, and width) in line 21–23 set additional layout properties.

One can further increase the space for the bars by reducing the space for the remaining vertical axis labels for the sector field.

https://www.nytimes.com/interactive/2020/07/30/upshot/coronavirus-stimulus-bill.html

One can reduce the space for axis labels by truncating or wrapping. To maintain the same amount of information, the text wrapping rule in line 24–26 changes the width of axis labels as `100`px. In the desktop version, the mark labels for the first rows have units ($00 billion). However, the partial transpose earlier (line 5–9) makes the unit repeat unnecessarily. To address the repeating labels, the modification rule in line 27–30 selects `mark.label`s for the `amount` value of `[200,0]` using the `data` keyword and changes the `format` as `",d"` (the format expression of d3.js [1]). Lastly, the mark label for the `amount` value of 1,010 ("$1,010 billion") is positioned inside the mark, the size of which is reduced after the earlier resizing and re-layout transformations. To reposition the mark label with text wrapping, the modification rule in line 31–34 queries the label similarly to the previous rule, and the `option` defines the layout properties (`anchor`, `dx`, and `width`).

3 PRINCIPLES FOR OUR CICERO COMPILER

Below, we provide a summary of principles for our Cicero compiler for reference. Details are described in the paper.

- **Associated elements**
  - **(P1)** Detect associated elements depending on how a user has defined the original design.
  - **(P2)** A transformation affecting the layout of a series of elements, such as adding, removing, or repositioning, has a downstream effect on the layout of their associated elements, but not the static style.
- **Default behaviours**
  - **(P3)** When adding a new element to a series of elements, its appearance should mimic the existing elements in the series.
  - **(P4)** Consider the appearance of elements in a similar role for new elements that are not part of an existing series of elements.
  - **(P5)** When there are multiple series of existing elements, select the one with the most similar structure.
  - **(P6)** Use the default options of the rendering grammar's compiler for newly added elements.
    - Place (new) externalized annotations below the chart.
    - Place (new) internalized data annotations (or mark labels) at the center or the bottom of the associated data mark.
    - Place (new) internalized non-data annotations at the center of the largest contiguous empty space in the chart.
- **Conflict management**
  - **(P7)** Apply the current rule to a view that has been transformed by the previous rules.
  - **(P8)** When there are two rules making changes to the same element for the same property, apply the last declared rule.
  - **(P9)** Assign higher priority to a more specific rule than a more generic rule for the same element.
  - **(P10)** Rules with the `important` property set to `true` have higher priorities than others.

REFERENCES

[1] Mike Bostock. 2015. d3-format. https://github.com/d3/d3-format Last accessed Sept 4, 2021.