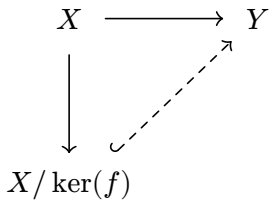
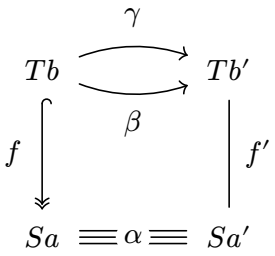
arrow-diagrams

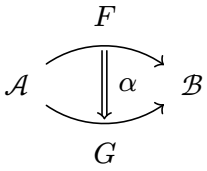
A Typst package for drawing commutative diagrams.

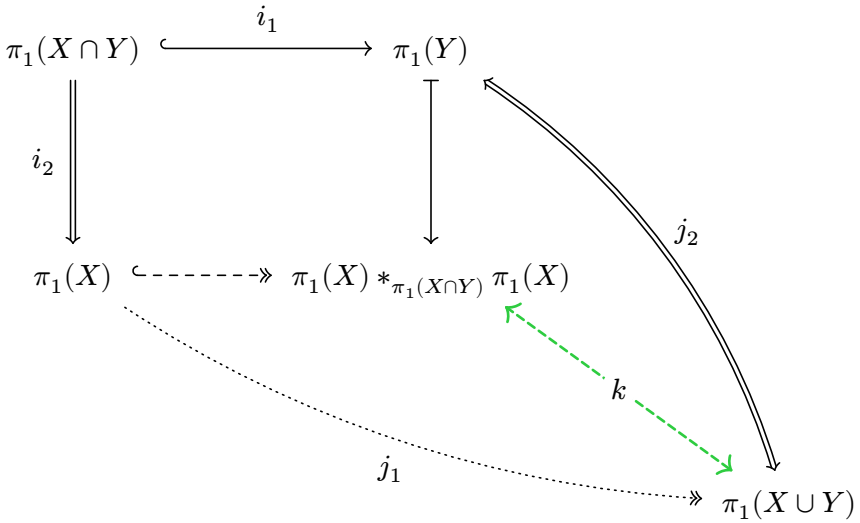
Contents

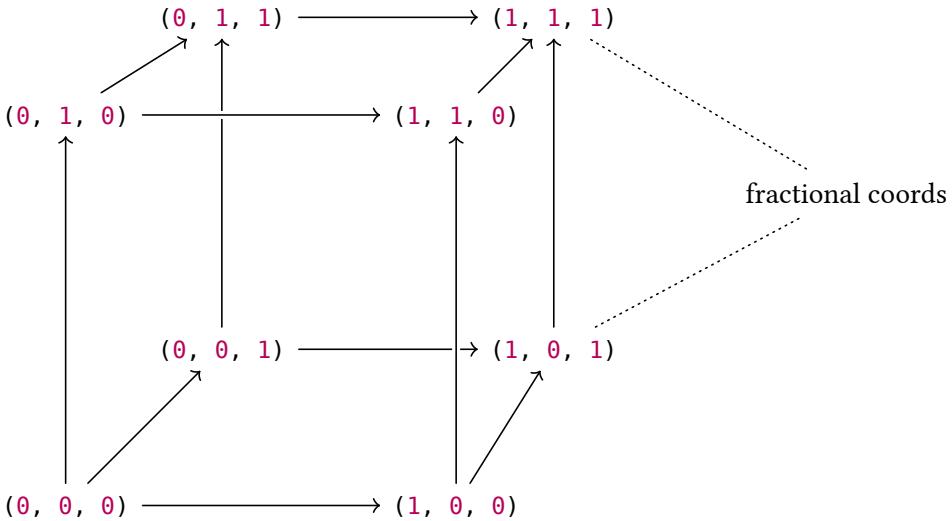
Examples


X
Y
X/ker(f)


γ
Tb
Tb′
β
f
f′
Sa
α
Sa′


F
𝒜
α
ℬ
G


i1
π1(X ∩ Y)
π1(Y)
i2
j2
π1(X)
π1(X) *π1(X∩Y) π1(X)
k
j1
π1(X ∪ Y)


(0, 1, 1)
(1, 1, 1)
(0, 1, 0)
(1, 1, 0)
fractional coords
(0, 0, 1)
(1, 0, 1)
(0, 0, 0)
(1, 0, 0)

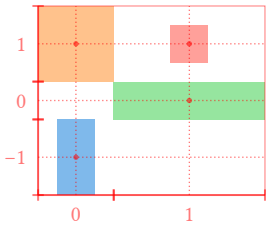
Tutorial

Layout

How the layouting works

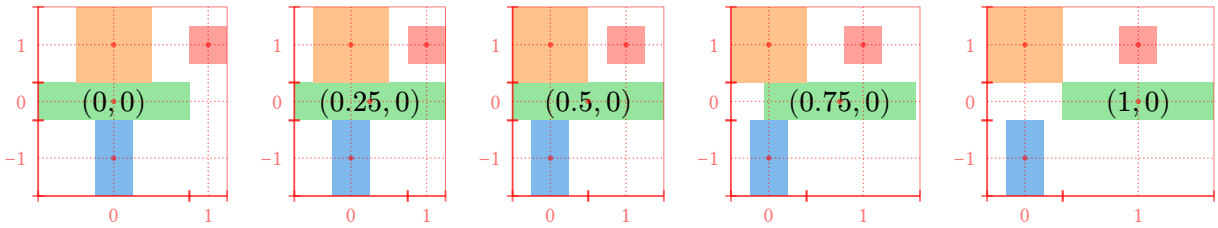
Each diagram is built on a grid of points, each at the center of a cell in a table layout. When a node is placed in a diagram, the rows and columns grow to accommodate the node's size.

This can be seen more clearly in diagrams with `debug: 1` and no cell padding:




```
#arrow-diagram(
  debug: 1,
  pad: 0pt,
  node((0,-1), box(fill: blue.lighten(50%),   width: 5mm,  height: 10mm)),
  node((1, 0), box(fill: green.lighten(50%),  width: 20mm, height:  5mm)),
  node((1, 1), box(fill: red.lighten(50%),    width:  5mm, height:  5mm)),
  node((0, 1), box(fill: orange.lighten(50%), width: 10mm, height: 10mm)),
)
```

While grid points are always at integer coordinates, nodes can also have **fractional coordinates**. A node between grid points still causes the neighbouring rows and columns to grow to accommodate its size, but only partially, depending on proximity. For example, notice how the column sizes change as the green box moves from $(0, 0)$ to $(1, 0)$:

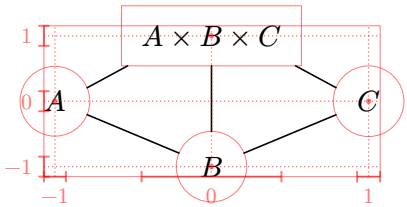


Specifically, fractional coordinates are dealt with by *linearly interpolating* the layout, in the sense that if a node is at $(0.25, 0)$, then the width of column $\lfloor 0.25 \rfloor = 0$ is at least 75% of the node's width, and column $\lceil 0.25 \rceil = 1$ at least 25% its width.

As a result, diagrams will automatically adjust when nodes grow or shrink, while still allowing you to place nodes at precise coordinates.

How connecting lines work

Lines between nodes connect to the node's bounding circle or bounding rectangle, depending on the node's aspect ratio.



The `defocus` correction

For aesthetic reasons, a line connecting to a node should not necessarily be focused to the node's exact center, especially if the node is short and wide or tall and narrow. Notice how in the figure above the lines connecting to the node $A \times B \times C$ would intersect slightly above its center, making the diagram look more comfortable. The effect of this is shown below:

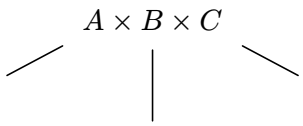


Figure 1: With defocus correction

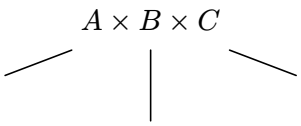
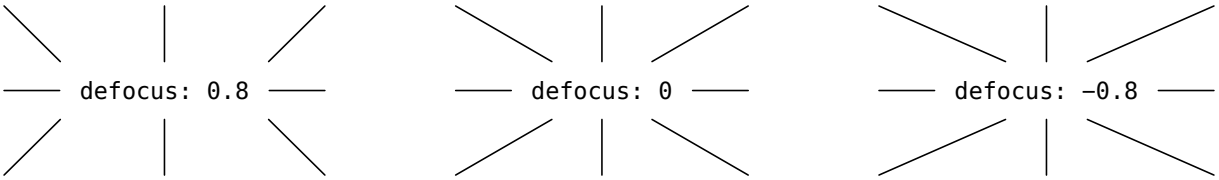


Figure 2: Without defocus correction

This correction is controlled by the `defocus` attribute of the node. It is best explained by example:



For `defocus: 0`, the connecting lines are directed exactly at the grid point at the node's center.

Function reference

compute-cells

Compute a lookup table of the attributes of each grid cell

Parameters

```
compute-cells(
  nodes: array,
  grid: dictionary,
  options
)
```

nodes `array`

Array of nodes to consider when calculating the sizes of cells, where each node of the form:

```
(
  pos: (i, j),
  size: (width, height),
)
```

grid `dictionary`

Grid specification of the form

```
(
  origin: (i, j),
  centers: ((x1, x2, ...), (y1, y2, ...)),
)
```

options

compute-grid

Determine the number, sizes and positions of rows and columns.

Parameters

```
compute-grid(
  nodes,
  options
)
```

nodes

options

expand-fractional-rects

Convert an array of rects with fractional positions into rects with integral positions.

A rect is a dictionary `(pos: (x, y), size: (width, height))`.

If a rect is centered at a factional position `floor(x) < x < ceil(x)`, it will be replaced by two new rects centered at `floor(x)` and `ceil(x)`. The total width of the original rect is split across the two new rects according two which one is closer. (E.g., if the original rect is at `x = 0.25`, the new rect at `x = 0` has 75% the original width and the rect at `x = 1` has 25%.) The same splitting procedure is done for `y` positions and heights.

Parameters

```
expand-fractional-rects(rects)
```

rects

round-arrow-cap-offset

Calculate cap offset of round-style arrow cap

Parameters

```
round-arrow-cap-offset(
  r: length,
  θ: angle,
  y: length
)
```

r `length`

Radius of curvature of arrow cap

θ `angle`

Angle made at the the arrow's vertex, from the central stroke line to the arrow's edge.

y `length`

Lateral offset from the central stroke line.

get-arc-connecting-points

Determine arc between two points with a given bend angle

The bend angle is the angle between chord of the arc (line connecting the points) and the tangent to the arc and the first point.

Parameters

```
get-arc-connecting-points(
  from: point,
  to: point,
  angle: angle
)
```

from `point`

2D vector of initial point.

to `point`

2D vector of final point.

angle `angle`

Bend angle.