NCSq Examples

Taichi Uemura

1. Basic usage

To draw a diagram, pass the `\NCSq.cd` command a rectangular array of cells.

```
\NCSq.cd(open NCSq in [
  [object ${A}; arrow right; object ${B}];
  [arrow down; empty; arrow down];
  [object ${C}; arrow right; object ${D}];
]);
```

$$\begin{array}{ccc} A & \longrightarrow & B \\ \downarrow & & \downarrow \\ C & \longrightarrow & D \end{array}$$

Here, `empty`, `object`, and `arrow` are functions defined in the `NCSq` module for building cells. The constant `empty` is the empty cell. The function `object math` creates an object (0-cell) with label `math`. The function `arrow direction` creates an arrow (1-cell) towards `direction`. To draw orthogonal arrows, set `direction` to be `left`, `right`, `up`, or `down`.

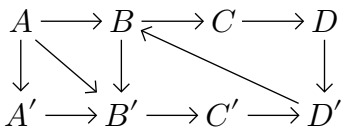
```
\NCSq.cd(open NCSq in [
  [object ${A}; arrow right; object ${B}];
  [arrow up; empty; arrow down];
  [object ${D}; arrow left; object ${C}];
]);
```

$$\begin{array}{ccc} A & \longrightarrow & B \\ \uparrow & & \downarrow \\ D & \longleftarrow & C \end{array}$$

To draw diagonal arrows, set `direction` to be `rd (m, n)`, `ru (m, n)`, `ld (m, n)`, or `lu`

`(m, n)`. Here, `rd`, `ru`, `ld`, and `lu` specifies rough direction of the arrow. `r`, `l`, `d`, and `u` stand for “right“, “left”, “down”, and “up”, respectively. `(m, n)`, where `m` and `n` are positive integers, determines the gradient of the arrow in numbers of cells.

```
\NCSq.cd(open NCSq in [
  [object ${A}; arrow right; object ${B};
   arrow right; object ${C}; arrow right; object ${D}];
  [arrow down; arrow (rd (1, 1)); arrow down;
   empty; arrow (lu (2, 1)); empty; arrow down];
  [object ${A'}; arrow right; object ${B'};
   arrow right; object ${C'}; arrow right; object ${D'}];
]);%
```

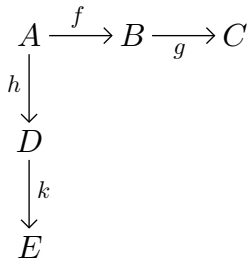


The function `arrow` accepts optional arguments to make labels.

```
arrow ?:left-label ?:right-label direction
```

`left-label` and `right-label` are labels on the left side and on the right side, respectively, facing the direction of an arrow.

```
\NCSq.cd(open NCSq in [
  [object ${A}; arrow ?:${f} right; object ${B};
   arrow ?*?:${g} right; object ${C}];
  [arrow ?*?:${h} down];
  [object ${D}];
  [arrow ?:${k} down];
  [object ${E}];
]);
```



The NCSq module also provides a math command \NCSq.cd-m which is suitable for drawing diagrams in math formulas.

```
\eqn($${
  \paren{
    \NCSq.cd-m!(open NCSq in [
      [object ${FA}; arrow ?:${Ff} right; object ${FB};
       arrow ?:${h} right; object ${C}];
    ])
  }' =
  \paren{
    \NCSq.cd-m!(open NCSq in [
      [object ${A}; arrow ?:${f} right; object ${B};
       arrow ?:${h'} right; object ${GC}];
    ])
  }
});
```

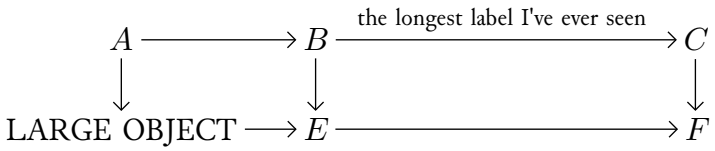
$$\left(FA \xrightarrow{Ff} FB \xrightarrow{h} C \right)' = \left(A \xrightarrow{f} B \xrightarrow{h'} GC \right)$$

1.1. Automatic arrow stretching

Arrows stretch automatically to match long labels and large objects.

```
\NCSq.cd(open NCSq in [
  [object ${A}; arrow right; object ${B};
```

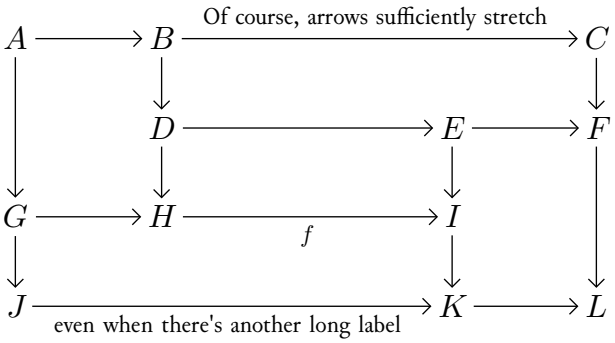
```
   arrow ?:${\text!{the longest label I've ever seen}} right;
   object ${C}];
  [arrow down; empty; arrow down;
   empty; arrow down];
  [object ${\text!{LARGE OBJECT}}; arrow right; object ${E};
   arrow right; object ${F}];
]);
```



The source and the target of an arrow are automatically detected: the source of an arrow is the nearest object in the opposite direction of the arrow; the target of an arrow is the nearest object in the direction of the arrow. If the source or the target of an arrow is not found, then the arrow will not be drawn.

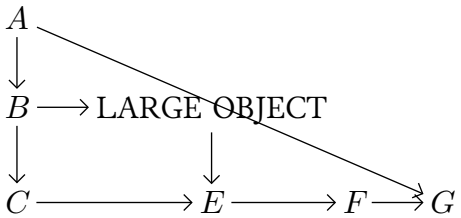
```
\NCSq.cd(open NCSq in [
  [object ${A}; arrow right; object ${B};
   arrow ?:${\text!{Of course, arrows sufficiently stretch}} right;
   empty; empty; object ${C}];
  [arrow down; empty; arrow down;
   empty; empty; empty; arrow down];
  [empty; empty; object ${D};
   arrow right; object ${E}; arrow right; object ${F}];
  [empty; empty; arrow down;
   empty; arrow down; empty; arrow down];
  [object ${G}; arrow right; object ${H};
   arrow ?*?:${f} right; object ${I}; empty; empty];
  [arrow down; empty; empty;
   empty; arrow down; empty; empty];
  [object ${J};
   arrow ?*?:${\text!{even when there's another long label}} right;
   empty; empty; object ${K}; arrow right; object ${L}];
```

```
]);
```



Note that, although the automatic arrow stretching prevents an arrow from overlapping with its source or target, an arrow may overlap with other objects and arrows, especially when using diagonal arrows.

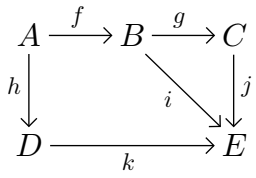
```
\NCSq.cd(open NCSq in [
  [object ${A}];
  [arrow down];
  [object ${B}; arrow right; object ${\text!{LARGE OBJECT}};
   arrow (rd (3, 2))];
  [arrow down; empty; arrow down];
  [object ${C}; arrow right; object ${E};
   arrow right; object ${F}; arrow right; object ${G}];
]);
```



1.2. The ncsq/aliases package

A helper package `ncsq/aliases` provides short aliases to functions like `object` and `arrow`.

```
\NCSq.cd(open NCSqAliases in [
  [o ${A}; a ?:${f} r; o ${B}; a ?:${g} r; o ${C}];
  [a ?*?:${h} d; e; e; a ?*?:${i} (rd (1, 1)); a ?:${j} d];
  [o ${D}; e; a ?*?:${k} r; e; o ${E}];
]);
```



1.3. The ncsq/easy-cd package

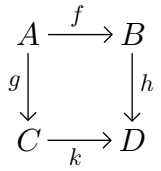
We often encounter a diagram, represented by a rectangular array $\left(c_{ij}\right)$ of cells, such that

- c_{ij} for i and j odd is an object;
- c_{ij} for i odd and j even is a right arrow;
- c_{ij} for i even and j odd is a down arrow;
- c_{ij} for i and j even is empty.

A helper package `ncsq/easy-cd` contains a module `EasyCD` which provides a command `\EasyCD.cd` to draw such diagrams with simplified syntax. The command `\EasyCD.cd` receives a rectangular array of math formulas and draws a diagram. For example,

```
\EasyCD.cd[
  ${| A | f | B |};
  ${| g |   | h |};
  ${| C | k | D |};
];
```

draws the following diagram.



A row `ms` at an odd position is thought of as a horizontal chain of arrows. A math formula `m` in `ms` at an odd position is converted to an object with label `m`. A math formula `m` in `ms` at an even position is converted to a right arrow with label `m`. The position of the label on the right arrow depends on the position of the row `ms`: if `ms` is in the upper half of the array, then the label is above the arrow; if `ms` is in the lower half of the array, then the label is below the arrow. For example, the first row in the above example is converted to

```
[object ${A}; arrow ?:${f} right; object ${B}]
```

while the last row is converted to

```
[object ${C}; arrow ?*?:${k} right; object ${D}]
```

A row `ms` at an even position contains vertical arrows. A math formula in `ms` at an even position is converted to the empty cell. A math formula `m` in `ms` at an odd position is converted to a down arrow with label `m`. The position of the label on the down arrow depends on the position of the math formula `m`: if `m` is in the left half of `ms`, then the label is on the left side of the arrow; if `m` is in the right half of `ms`, then the label is on the right side of the arrow. For example, the middle row in the above example is converted to

```
[arrow ?*?:${g} down; empty; arrow ?:${h} down]
```

Here's a larger example.

```
\EasyCD.cd[
  ${| A | a | B | b | C | c | D |};
  ${| d |   | e |   | f |   | g |};
```

```
  ${| E | h | F | i | G | j | H |};
  ${| k |   | l |   | m |   | n |};
  ${| I | o | J | p | K | q | L |};
];
```

$$\begin{array}{ccccc}
A & \xrightarrow{a} & B & \xrightarrow{b} & C & \xrightarrow{c} & D \\
\downarrow{\scriptstyle d} & & \downarrow{\scriptstyle e} & & \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle g} \\
E & \xrightarrow{h} & F & \xrightarrow{i} & G & \xrightarrow{j} & H \\
\downarrow{\scriptstyle k} & & \downarrow{\scriptstyle l} & & \downarrow{\scriptstyle m} & & \downarrow{\scriptstyle n} \\
I & \xrightarrow[o]{} & J & \xrightarrow[p]{} & K & \xrightarrow[q]{} & L
\end{array}$$

1.4. Parallel arrows

The `NCSq` module provides another function `stack` for building parallel arrows.

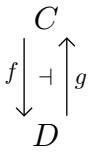
```
stack direction cells
```

`direction` can be `horizontal` or `vertical`, and `cells` is a list of cells.

```
\NCSq.cd(open NCSq in [
  [object ${A};
   stack vertical [
     arrow ?:${f} right;
     arrow ?*?:${g} right;
   ];
   object ${B}];
]);
```

$$A \underset{g}{\overset{f}{\rightrightarrows}} B$$

```
\NCSq.cd(open NCSq in [
  [object ${C}];
  [stack horizontal [
    arrow ?*?:${f} down;
    object ${\dashv};
    arrow ?*?:${g} up;
  ]];
  [object ${D}];
]);
```



In a horizontal (vertical) stack, only objects and vertical (horizontal, respectively) arrows are used, and other cells are ignored.

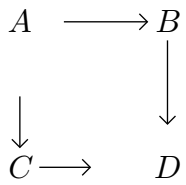
2. Changing styles

The function `object` accepts an optional argument to change paddings.

```
object ?:(pd-left, pd-right, pd-top, pd-bottom) label
```

Paddings are specified by ratios to the font size. The default value is `(pd-left, pd-right, pd-top, pd-bottom) = (0.25, 0.25, 0.25, 0.25)`.

```
\NCSq.cd(open NCSq in [
  [object ?:(0.25, 1.0, 0.25, 2.0) ${A};
   arrow right; object ${B}];
  [arrow down; empty; arrow down];
  [object ${C}; arrow right;
   object ?:(2.0, 0.25, 1.0, 3.0) ${D}];
]);
```

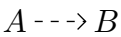


The function `arrow` accepts the third optional argument to change the style.

```
arrow ?:left-label ?:right-label ?:style direction
```

The `NCSq` module provides three styles: `solid` (default); `dashed`; and `equal`.

```
\NCSq.cd(open NCSq in [
  [object ${A}; arrow ?*?*?:dashed right; object ${B}];
]);
```



```
\NCSq.cd(open NCSq in [
  [object ${A}; arrow ?*?*?:equal right; object ${B}];
]);
```

The `NCSq` module also provides a function `eqarrow` for building an equality arrow. `eqarrow direction` is simply an alias to `arrow ?*?*?:equal direction`.

```
\NCSq.cd(open NCSq in [
  [object ${A}; eqarrow right; object ${B}];
]);
```

$$A = B$$

See Section 2.1 for deeper configuration.

2.1. Arrow styles

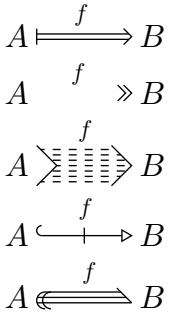
The `ncsq/arrows` package, which is loaded by `ncsq/ncsq`, provides several arrow styles. An arrow style consists of three components: body; tail; and head. An arrow style is constructed by `NCSqArrowStyle.of-bth` function.

```
NCSqArrowStyle.of-bth body tail head
```

Modules `NCSqArrowBody`, `NCSqArrowTail` and `NCSqArrowHead` provide a few body styles, tail styles and head styles, respectively, and those styles can be used in any combination.

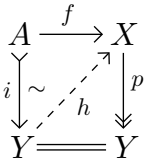
```
\NCSq.cd(
  let style-1 = NCSqArrowStyle.of-bth
  NCSqArrowBody.solid-2 NCSqArrowTail.vdash NCSqArrowHead.vee in
  let style-2 = NCSqArrowStyle.of-bth
  NCSqArrowBody.none NCSqArrowTail.none NCSqArrowHead.vee-2 in
  let style-3 = NCSqArrowStyle.of-bth
  (NCSqArrowBody.multi-dashed 7)
  NCSqArrowTail.vee NCSqArrowHead.vee in
  let style-4 = NCSqArrowStyle.of-bth
  NCSqArrowBody.(cross solid)
  NCSqArrowTail.hook-l NCSqArrowHead.triangle in
  let style-5 = NCSqArrowStyle.of-bth
  (NCSqArrowBody.multi-solid 3)
  NCSqArrowTail.hook-r-2 NCSqArrowHead.harpoon-l in
  open NCSq in
  let row style = [
    object ${A}; arrow ?:${f}?*?:style right; object ${B}
  ] in [
    row style-1;
    row style-2;
    row style-3;
```

```
    row style-4;
    row style-5;
  ]
);
```



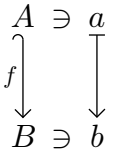
The `NCSqArrowStyle` module provides preconfigured arrow styles.

```
\NCSq.cd(open NCSq in open NCSqArrowStyle in [
  [object ${A}; arrow ?:${f} right; object ${X}];
  [arrow ?:${\sim}?:${i}?:tail down;
   arrow ?*?:${h}?:dashed (ru (1, 1));
   arrow ?:${p}?*?:two-heads down];
  [object ${Y}; arrow ?*?*?:equal right;
   object ${Y}];
]);
```




```
\NCSq.cd(open NCSq in open NCSqArrowStyle in [
  [object ${A}; object ${\ni}; object ${a}];
  [arrow ?*?:${f}?:(hook-r) down; empty;
   arrow ?*?*?:(mapsto) down];
```

```
  [object ${B}; object ${\ni}; object ${b}];
]);
```



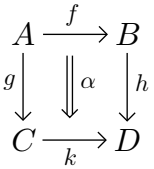
NCSq uses `satysfi-arrows` package. See the documentation of that package to define more arrow styles.

3. Advanced usage

3.1. Invisible sources and targets

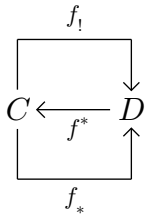
The `NCSq` module provides a constant cell `point`. Like the `empty` cell, it draws nothing, but it can be the source and the target of an arrow. This allows us, for example, to draw arrows between arrows.

```
\NCSq.cd(open NCSq in open NCSqArrowStyle in [
  [object ${A}; arrow ?:${f} right; object ${B}];
  [empty; point; empty];
  [arrow ?*?:${g} down; arrow ?:${\alpha}?*?:solid-2 down;
   arrow ?:${h} down];
  [empty; point; empty];
  [object ${C}; arrow ?*?:${k} right; object ${D}];
]);
```



Another usage is to draw polylines.

```
\NCSq.cd(open NCSq in open NCSqArrowStyle in [
  [point; arrow ?:${f_{\!}}?*?:line right; point];
  [arrow ?*?*?:line up; empty; arrow down];
  [object ${C}; arrow ?:${f^{\ast}}left; object ${D}];
  [arrow ?*?*?:line down; empty; arrow up];
  [point; arrow ?*?:${f_{\ast}}?:line right; point];
]);
```



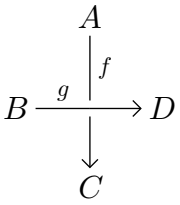
3.2. Crossed arrows

The `NCSq` module provides a function `crossing`.

```
crossing ?:padding ?:style direction
```

`direction` can be `horizontal` or `vertical`. Like `point`, it draws nothing and can be the source and the target of an arrow. The difference is that `crossing direction` has paddings in the direction orthogonal to `direction`. The intended usage is as follows.

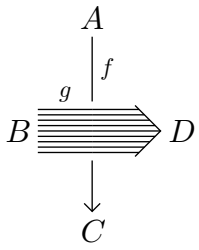
```
\NCSq.cd(open NCSq in open NCSqArrowStyle in [
  [empty; empty; object ${A}];
  [empty; empty; arrow ?:${f}?*?:line down];
  [object ${B}; arrow ?:${g}?*?:line right;
   crossing horizontal; arrow right; object ${D}];
  [empty; empty; arrow down];
  [empty; empty; object ${C}];
]);
```



The central cell `crossing horizontal` here is the target of the horizontal line with label g, the source of the right arrow, the target of the vertical line with label f, and the source of the down arrow. Because the central cell does not have horizontal paddings, the horizontal line and the right arrow look like a single horizontal arrow. On the other hand, since the central cell does have vertical paddings, the vertical line and the down arrow are separated. Therefore, the whole diagram looks like the right arrow is crossing over the down arrow.

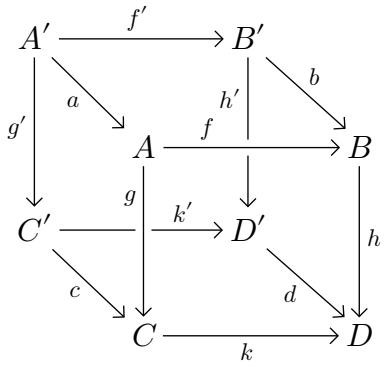
The optional argument `style` tells the cell the width of the crossing arrow.

```
\NCSq.cd(
  open NCSq in open NCSqArrowStyle in
  let bs = NCSqArrowBody.multi-solid 9 in
  let sty1 = of-bth bs NCSqArrowTail.none NCSqArrowHead.none in
  let sty2 = of-bth bs NCSqArrowTail.none NCSqArrowHead.vee in
  [
    [empty; empty; object ${A}];
    [empty; empty; arrow ?:${f}?*?:line down];
    [object ${B}; arrow ?:${g}?*?:sty1 right;
     crossing ?*?:sty2 horizontal;
     arrow ?*?*?:sty2 right; object ${D}];
    [empty; empty; arrow down];
    [empty; empty; object ${C}];
  ]
);%
```



A cube is a good example of a diagram with crossed arrows.

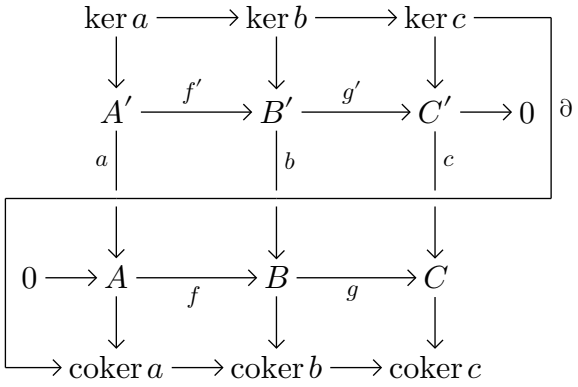
```
\NCSq.cd(open NCSq in open NCSqArrowStyle in [
  [object ${A'}; empty; arrow ?:${f'} right;
   empty; object ${B'}; empty; empty];
  [empty; arrow ?*?:${a} (rd (1, 1)); empty;
   empty; arrow ?*?:${h'}?:line down;
   arrow ?:${b} (rd (1, 1)); empty];
  [arrow ?*?:${g'} down; empty; object ${A};
   arrow ?:${f}?*?:line right; crossing horizontal;
   arrow right; object ${B}];
  [empty; empty; arrow ?*?:${g}?:line down;
   empty; arrow down; empty; empty];
  [object ${C'}; arrow ?*?*?:line right;
   crossing vertical; arrow ?:${k'} right;
   object ${D'}; empty; arrow ?:${h} down];
  [empty; arrow ?*?:${c} (rd (1, 1)); arrow down;
   empty; empty; arrow ?*?:${d} (rd (1, 1)); empty];
  [empty; empty; object ${C}; empty;
   arrow ?*?:${k} right; empty; object ${D}];
]);%
```



Now it's time to state the snake lemma.

```
\NCSq.cd(
  open NCSq in open NCSqArrowStyle in
  let-math \ker = math-char MathOp `ker` in
  let-math \coker = math-char MathOp `coker` in
  [
    [empty; empty; empty; object ${\ker a};
     arrow right; object ${\ker b}; arrow right;
     object ${\ker c}; arrow ?*?*?:line right;
     empty; point];
    [empty; empty; empty; arrow down;
     empty; arrow down; empty; arrow down;
     empty; empty; arrow ?:${\partial}?*?:line down];
    [empty; empty; empty; object ${A'};
     arrow ?:${f'} right; object ${B'};
     arrow ?:${g'} right; object ${C'};
     arrow right; object ${0}];
    [empty; empty; empty; arrow ?*?:${a}?:line down;
     empty; arrow ?:${b}?*?:line down; empty;
     arrow ?:${c}?*?:line down];
    [point; arrow ?*?*?:line left; empty;
     crossing horizontal; arrow ?*?*?:line left;
     crossing horizontal; arrow ?*?*?:line left;
     crossing horizontal; arrow ?*?*?:line left;
```

```
     empty; point];
    [empty; empty; empty; arrow down;
     empty; arrow down; empty; arrow down];
    [arrow ?*?*?:line down; object ${0}; arrow right;
     object ${A}; arrow ?*?:${f} right; object ${B};
     arrow ?*?:${g} right; object ${C}];
    [empty; empty; empty; arrow down; empty;
     arrow down; empty; arrow down];
    [point; arrow right; empty; object ${\coker a};
     arrow right; object ${\coker b};
     arrow right; object ${\coker c};];
  ]
);
```



4. Utilities

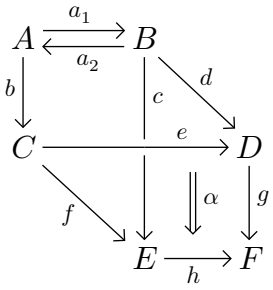
The `NCSq` module provides some utilities for building diagrams.

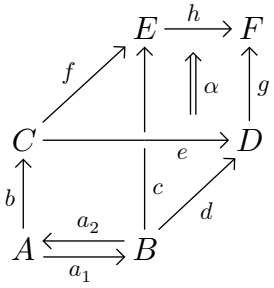
The `invert-v` function inverts a diagram vertically.

```
% In preamble
let cellss1 = open NCSq in open NCSqArrowStyle in [
  [object ${A}; stack vertical [
```

```
    arrow ?:${a_{1}} right;
    arrow ?:${a_{2}} left;
  ]; object ${B}];
  [arrow ?*?:${b} down; empty;
   arrow ?:${c}?*?:line down; arrow ?:${d} (rd (1, 1))];
  [object ${C}; arrow ?*?*?:line right;
   crossing horizontal; arrow ?:${e} right; object ${D}];
  [empty; empty; empty; point; empty];
  [empty; arrow ?*?:${f} (rd (1, 2)); arrow down;
   arrow ?:${\alpha}?*?:solid-2 down;
   arrow ?:${g} down];
  [empty; empty; empty; point; empty];
  [empty; empty; object ${E};
   arrow ?*?:${h} right; object ${F}];
]
```

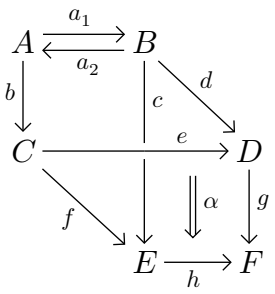
```
\math-list(
  ${| \NCSq.cd-m!(cellss1) | \NCSq.cd-m!(NCSq.invert-v cellss1) |}
);
```

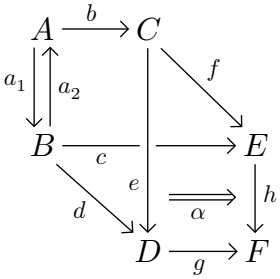




The `transpose` function flips a diagram over its diagonal.

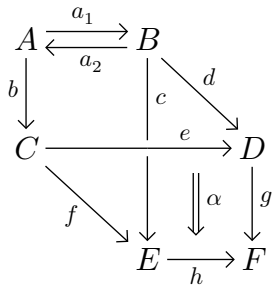
```
\math-list(
  ${| \NCSq.cd-m!(cellss1) | \NCSq.cd-m!(NCSq.transpose cellss1) |}
);
```

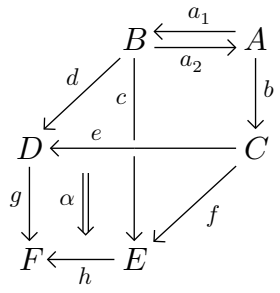




Using the functions `invert-v` and `transpose`, one can invert a diagram horizontally, but why not preconfigure a function `invert-h`.

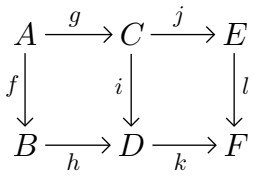
```
\math-list(
  ${| \NCSq.cd-m!(cellss1) | \NCSq.cd-m!(NCSq.invert-h cellss1) |}
);
```





The `EasyCD` module also provides the `transpose` function.

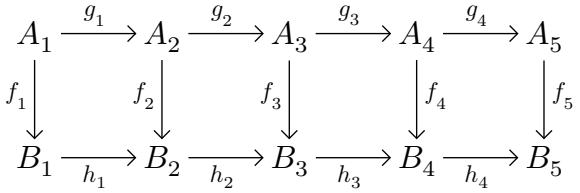
```
\EasyCD.cd(EasyCD.transpose [
  ${| A | f | B |};
  ${| g |   | h |};
  ${| C | i | D |};
  ${| j |   | k |};
  ${| E | l | F |};
]);
```



5. More examples

Incomplete diagrams (diagrams that contain arrows missing source or target) may be useful for building diagrams programmatically.

```
\EasyCD.cd(
  % Incomplete square (one face is missing)
  let square-from a f b g h = [
    ${| #a | #f | #b |};
    ${| #g |    | #h |};
  ] in
  [1; 2; 3; 4; 5]
  |> List.map (fun i -> (
    let mi = math-char MathOrd (arabic i) in
    square-from ${A_{#mi}} ${f_{#mi}} ${B_{#mi}}
    ${g_{#mi}} ${h_{#mi}}))
  |> List.concat
  |> EasyCD.transpose
);
```



Arrows nicely stretch vertically too, though one might rarely write such tall labels on arrows.

```
\NCSq.cd(open NCSq in [
  [object ${A}; arrow right; object ${B}];
  [arrow ?*?:${\frac{\frac{\frac{a}{b}}{\frac{c}{d}}}
              {\frac{\frac{e}{f}}{\frac{g}{h}}}} down;
   empty; arrow down];
  [empty; empty; object ${C}];
  [empty; empty;
   arrow ?:${\frac{\frac{\frac{i}{j}}{\frac{k}{l}}}
            {\frac{\frac{m}{n}}{\frac{o}{p}}}} down];
  [object ${D}; arrow right; empty];
  [arrow down; empty; empty];
  [object ${E}; arrow right; object ${F}];
]);
```