

String diagrams for higher mathematics with `wiggle.py`

Simon Burton

April 20th, 2023

simon.burton@quantinuum.com

Outline

(1) String diagrams for higher mathematics

(2) with `wiggle.py`

Outline

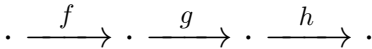
- (1) String diagrams for higher mathematics
 - ~~(1a) all about associativity, unitality, etc.~~
- (2) with `wiggle.py`

Outline

- (1) String diagrams for higher mathematics
 - ~~(1a) all about associativity, unitality, etc.~~
- (2) with `wiggle.py`
 - (2a) demo
 - (2b) implementation notes

A monoid

“... and then ...”



Arrow diagram



String diagram

A category

“Multi-coloured monoid”

$$A \xrightarrow{f} B \xrightarrow{g} C \xrightarrow{h} D$$



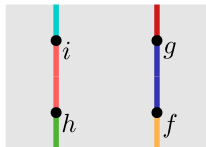
Arrow diagram

String diagram

A monoidal category

“... and then ..., but also...”

$$A \otimes B \xrightarrow{h \otimes f} C \otimes D \xrightarrow{i \otimes g} E \otimes F$$



Arrow diagram

String diagram

A bicategory

“multi-coloured monoidal category”

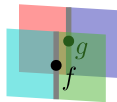


Arrow diagram String diagram

A monoidal bicategory

“... and then ..., but also..., meanwhile...”

$$m \otimes o \begin{array}{c} \leftarrow \\ \xrightarrow{f \otimes g} \\ \leftarrow \end{array} l \otimes n$$

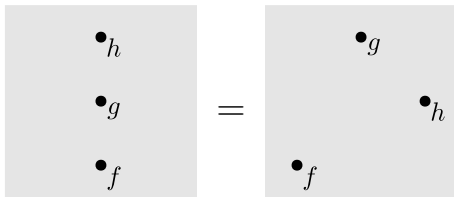


Arrow diagram

String diagram

Commutativity

from “and then” to “and”



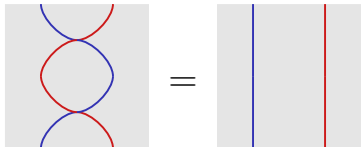
Commutative monoids

Higher commutativity







Equations become morphisms:



That satisfy further equations:







Higher equality

In a given 2-category, an *equivalence* between objects  and  is a pair of 1-cells  ,  and isomorphisms  ,  :



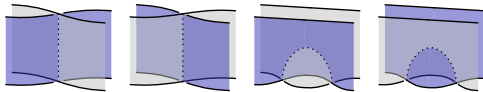
Adjointness

An *adjoint pair* of 1-cells is a *left adjoint*  and a *right adjoint*  such that there exists 2-cells  and  called the *unit* and *counit* respectively, with

$$\begin{array}{|c|} \hline \text{red} \\ \hline \end{array} \begin{array}{|c|} \hline \text{grey} \\ \hline \end{array} \begin{array}{c} \uparrow \\ \text{S-curve} \\ \uparrow \end{array} = \begin{array}{|c|} \hline \text{grey} \\ \hline \end{array} \begin{array}{|c|} \hline \text{red} \\ \hline \end{array} \begin{array}{c} \uparrow \end{array}, \quad \begin{array}{|c|} \hline \text{red} \\ \hline \end{array} \begin{array}{|c|} \hline \text{grey} \\ \hline \end{array} \begin{array}{c} \downarrow \end{array} = \begin{array}{|c|} \hline \text{red} \\ \hline \end{array} \begin{array}{|c|} \hline \text{grey} \\ \hline \end{array} \begin{array}{c} \downarrow \\ \text{S-curve} \\ \downarrow \end{array}.$$

Higher commutativity

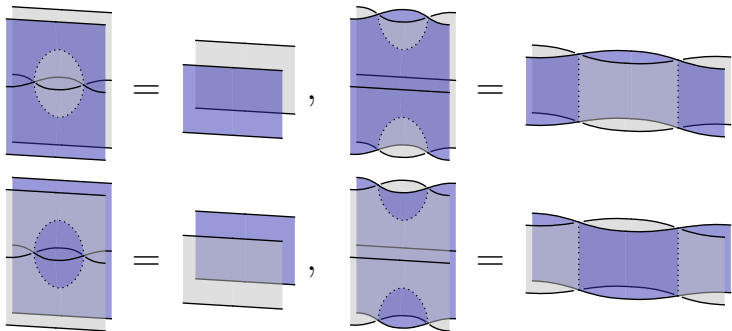
Braided monoidal bicategory has:



& these form an adjoint equivalence...

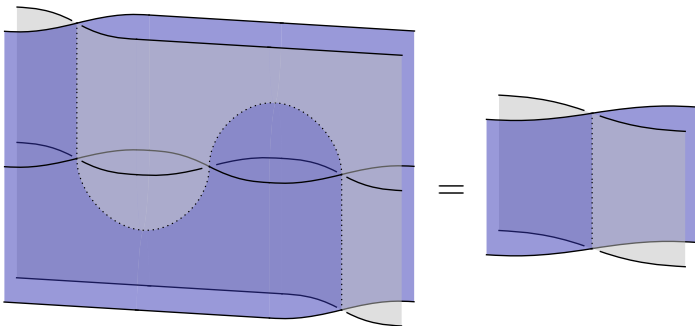
Higher commutativity

Equivalence:



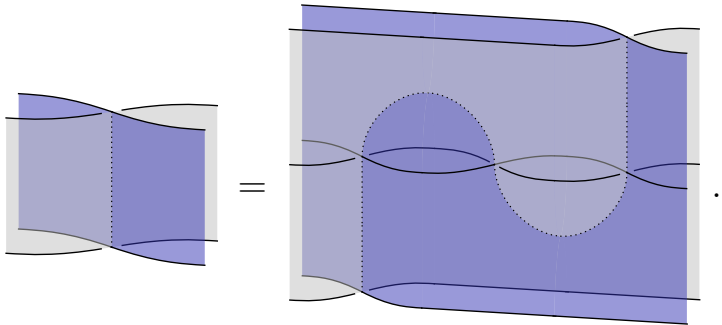
Higher commutativity

Adjointness:



Higher commutativity

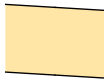
Adjointness:



demo of wiggle.py

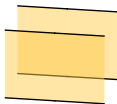
demo of wiggle.py

```
from huygens.namespace import color
from huygens.wiggle import Cell0, Cell1, Cell2
yellow = color.rgb(1.0, 0.80, 0.3, 0.5)
Cell0("m", fill=yellow)
```



demo of wiggle.py

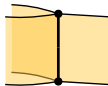
```
from huygens.namespace import color
from huygens.wiggle import Cell0, Cell1, Cell2
yellow = color.rgb(1.0, 0.80, 0.3, 0.5)
m = Cell0("m", fill=yellow)
m@m
```



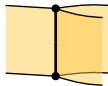
demo of wiggle.py

```
m = Cell0("m", fill=yellow)
```

```
Cell1(m@m, m)
```



```
Cell1(m, m@m)
```



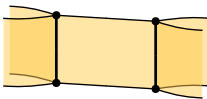
demo of wigggle.py

```
m = Cell10("m", fill=yellow)
```

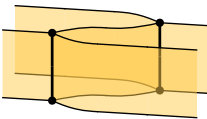
```
mm_m = Cell11(m@m, m)
```

```
m_mm = Cell11(m, m@m)
```

```
mm_m << m_mm
```

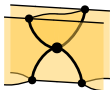


```
(m_mm @ m) << (m @ mm_m)
```



demo of wiggle.py

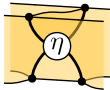
```
m = Cell0("m", fill=yellow)
mm_m = Cell1(m@m, m)
m_mm = Cell1(m, m@m)
src = mm_m << m_mm
tgt = (m_mm @ m) << (m @ mm_m)
Cell2(tgt, src)
```



demo of wiggle.py

...

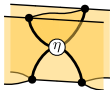
```
from huygens.namespace import *  
p = path.circle(0, 0, 0.2)  
cvs = Canvas()  
cvs.fill(p, [white]).stroke(p).  
cvs.text(0, 0, r"$\eta$", st_center)  
Cell2(tgt, src, pip_cvs=cvs)
```



demo of wiggle.py

...

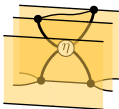
```
from Huygens.namespace import *  
p = path.circle(0, 0, 0.2)  
cvs = Canvas().scale(0.6)  
cvs.fill(p, [white]).stroke(p).  
cvs.text(0, 0, r"$\eta$", st_center)  
Cell2(tgt, src, pip_cvs=cvs)
```



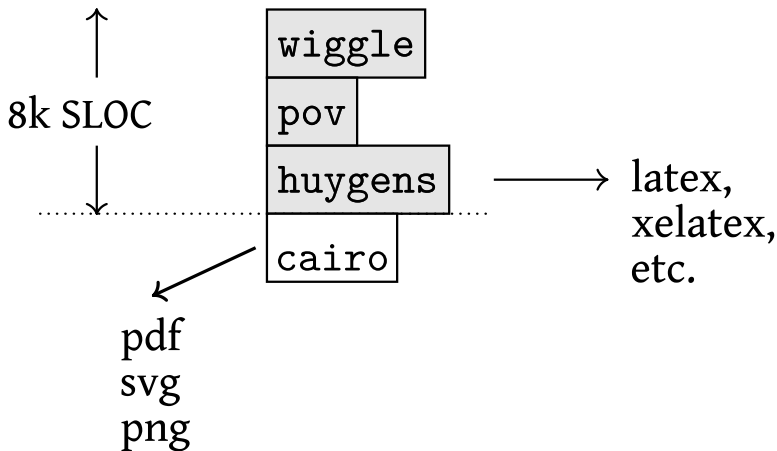
demo of wigggle.py

...

```
from Huygens.namespace import *  
p = path.circle(0, 0, 0.2)  
cvs = Canvas().scale(0.6)  
cvs.fill(p, [white]).stroke(p).  
cvs.text(0, 0, r"$\eta$", st_center)  
m @ Cell2(tgt, src, pip_cvs=cvs)
```



Implementation notes



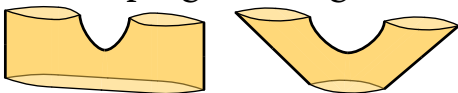
Implementation notes

- reasonable API
 - instance re-use via clones

```
m = Cell10("m", fill=yellow)
```

```
m@m
```

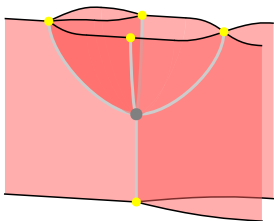
- linear programming + user constraints



- a bazillion parameters; customizable defaults

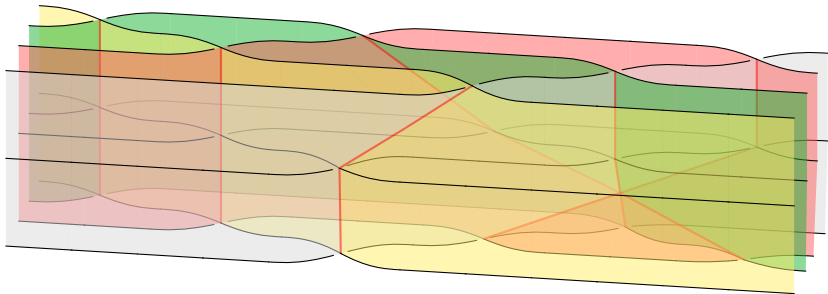
Implementation notes

- vector 3d, not raster 3d
 - alpha blending
 - back to front render order
 - render cones from curvy triangles



Implementation notes

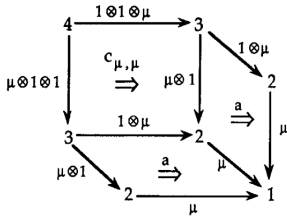
- tree-like data structure
 - lateral, horizontal and vertical composition



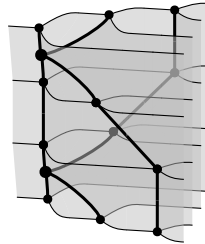
Implementation notes

- API stability
- more documentation
- edge-cases, work-arounds, bugs, ...
 - fixable in principle

The end



How it started



How its going