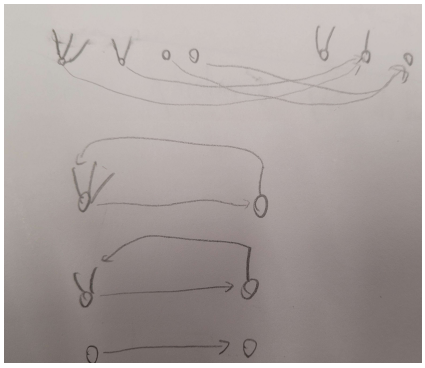
PolyMapBasic.jl

by Thomas Purdy, help from Lauresa Stilling.

I wrote a small Julia library, PolyMapBasic.jl which does a few things with polynomial functors. It makes use of Catlab's schema categories, GraphViz, and Poly.jl.

- FinPolyMap, a data structure representing a map between two polynomials.
- GraphPoly, a function which can graph a FinPoly
- Definitions for $\curlyvee$ and $\triangleleft$, defined in terms of $+$, $*$ and $\otimes$ from Poly.jl.

Code can be seen here: https://github.com/neonWhiteout/PolyMapBasic



Is represented by:

```
julia> srcposa = [3,2,0,0]; tgtposa = [2, 1, 0]; posmap = [2,2,3,3]; dirmap = [2 => [1], 2 => [2], 3 => Vector{Int}(), 3 => Vector{Int}()];

julia> FinPolyMap(srcposa, tgtposa, posmap, dirmap)
FinPolyMap {SrcPos:4, SrcDir:5, TgtPos:3, TgtDir:3, φ♯:2}
```

SrcPos	φ
1	2
2	2
3	3
4	3

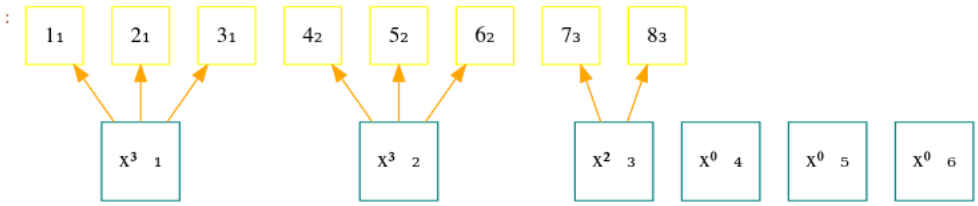
SrcDir	srcpoly
1	1
2	1
3	1
4	2
5	2

TgtDir	tgtpoly
1	1
2	1
3	2

φ♯	φ♯sp	φ♯td	φ♯sd
1	1	3	1
2	2	3	5

Graphing a FinPoly as a corolla:

```
: 1 GraphPoly(FinPoly([3,3,2,0,0,0]) ; charval = 'x')
```



Tri operator on two polys:

```
julia> FinPoly([2]) ◁ FinPoly([3,1])
FinPoly {Pos:4, Dir:16}
```

Dir	pos
1	1
2	1
3	1
4	2
5	3
6	3
7	3
8	4
9	1
10	2
11	1
12	2
13	1
14	2
15	3
16	4

Further areas of development would include defining functions on mappings, allowing mappings to act as Moore machines and move between states, graphing mappings.

There should also be more assert statements, to ensure malformed input isn't accepted.