Propagators: An Introduction

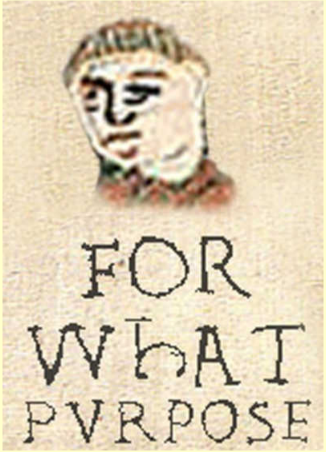
George Wilson

Data61/CSIRO

george.wilson@data61.csiro.au

November 12, 2017

What?



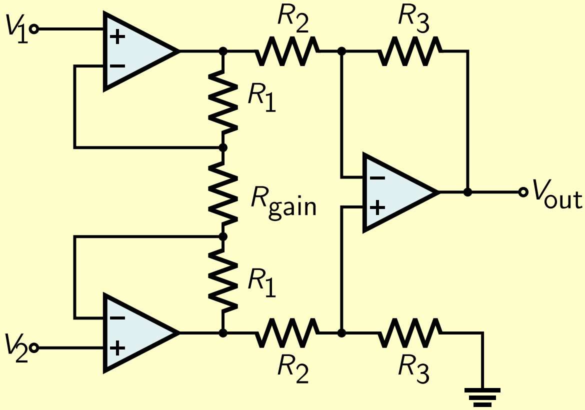
Why?

Beginnings as early as the 1970's at MIT

- Guy L. Steele Jr.
- Gerald J. Sussman
- Richard Stallman

More recently:

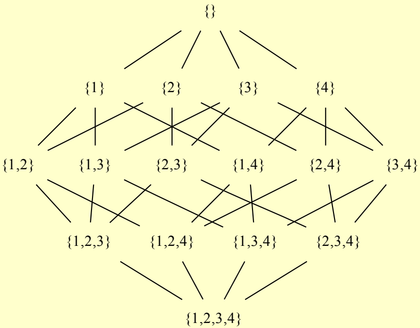
- Alexey Radul



```
(define (map f xs)
  (cond ((null? xs) '())
        (else (cons (f (car xs))
                    (map f (cdr xs))))))
```

And then

- Edward Kmett



$$x \leq y \implies f(x) \leq f(y)$$

Propagators

The *propagator model* is a model of computation
We model computations as *propagator networks*

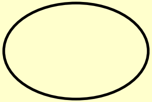
The *propagator model* is a model of computation
We model computations as *propagator networks*

Propagator networks:

- are extremely expressive
- lend themselves to parallel and distributed evaluation
- allow different strategies of problem-solving to seamlessly cooperate

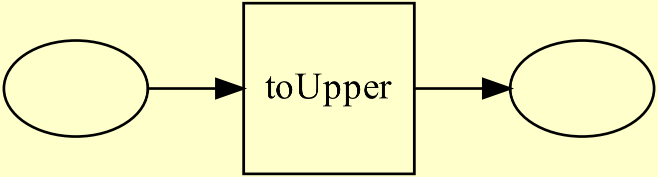
A propagator network comprises

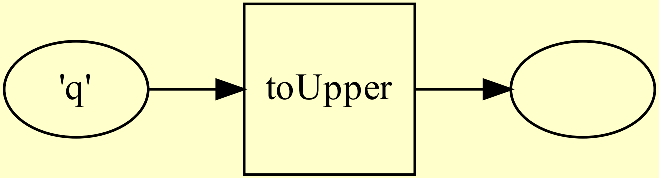
- cells
- propagators
- connections between cells and propagators

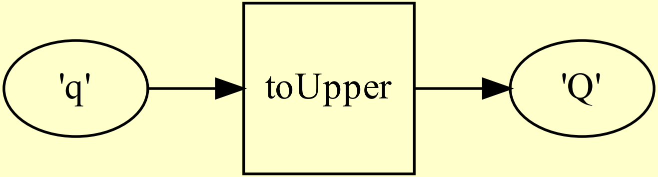


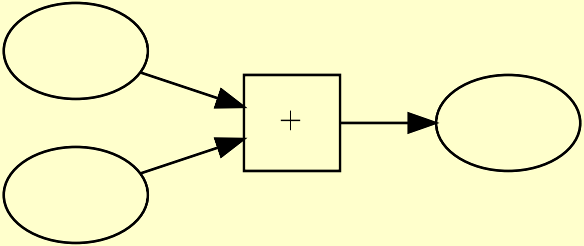
3

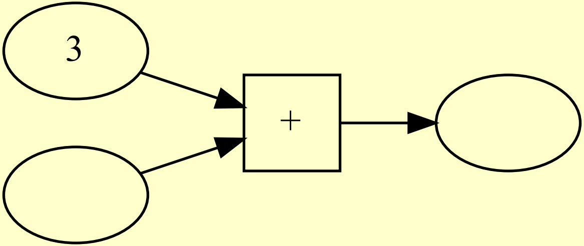
toUpper

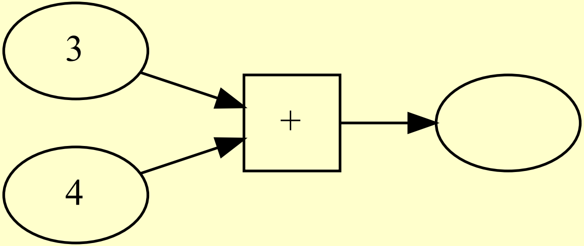

toUpper

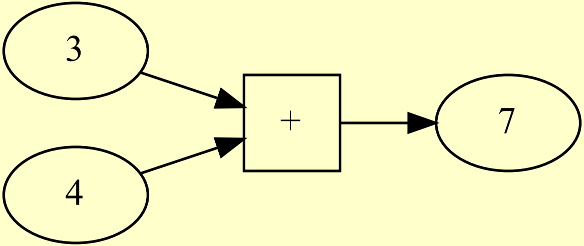

'q'
toUpper


'q'
toUpper
'Q'




3
+


3
4
+


3
4
+
7

$$z \leftarrow x + y$$

$$z = x + y$$

$$7 = x + 4$$

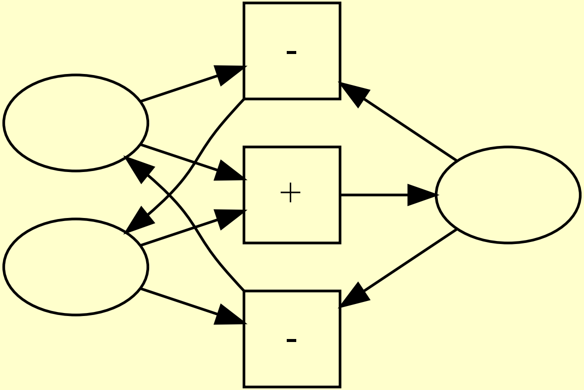
$$7 = 3 + 4$$

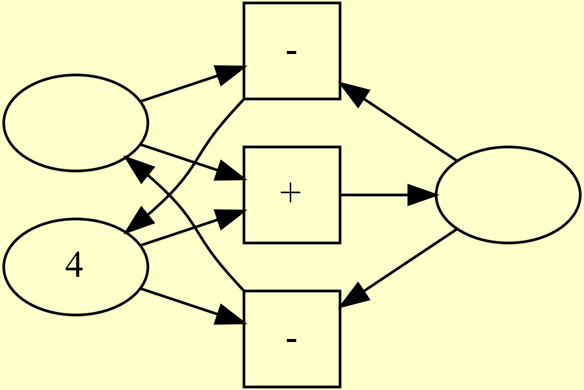
$$z = x + y$$

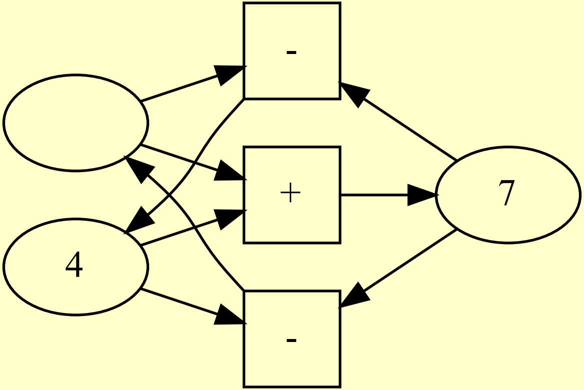
$$z \leftarrow x + y$$

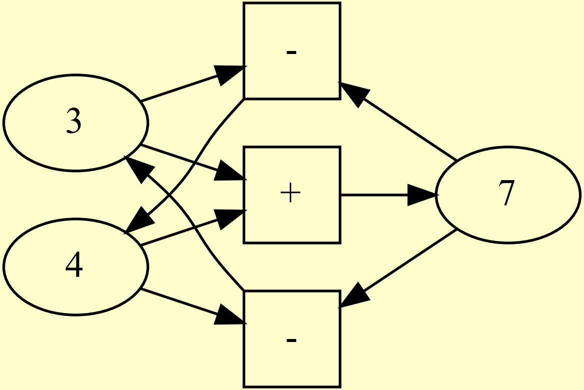
$$x \leftarrow z - y$$

$$y \leftarrow z - x$$




-
+
-
4

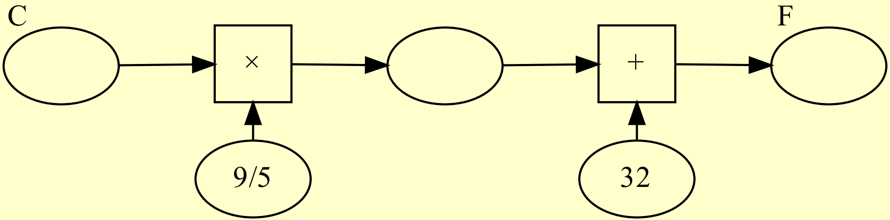

-
+
-
4
7



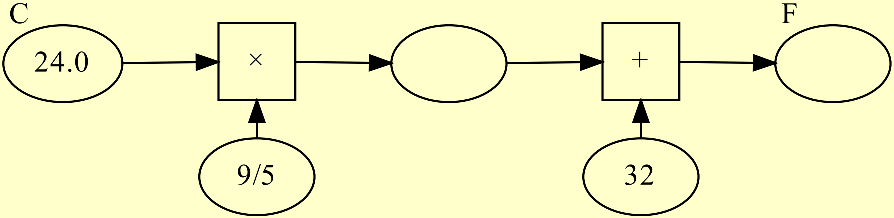
-
3
+
7
4
-

Propagators let us express multi-directional relationships!

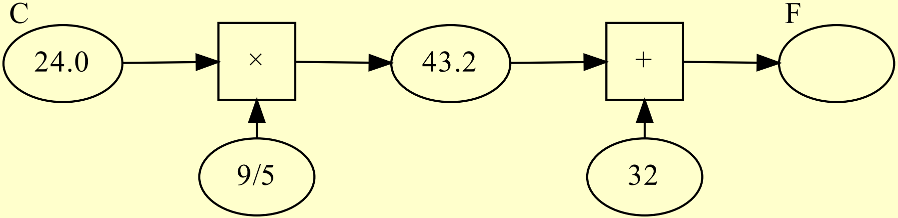
$$^{\circ}F = {^{\circ}C} \times \frac{9}{5} + 32$$



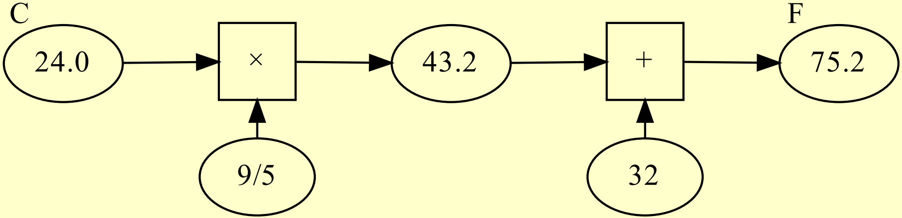
$$°F = °C \times \frac{9}{5} + 32$$



$$°F = °C \times \frac{9}{5} + 32$$

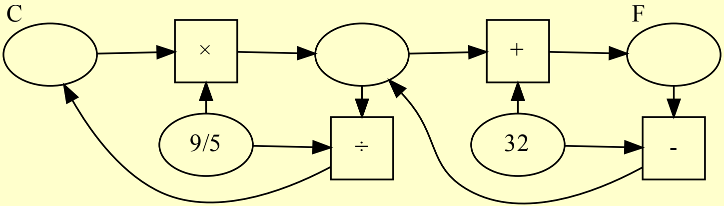


$$°F = °C \times \frac{9}{5} + 32$$



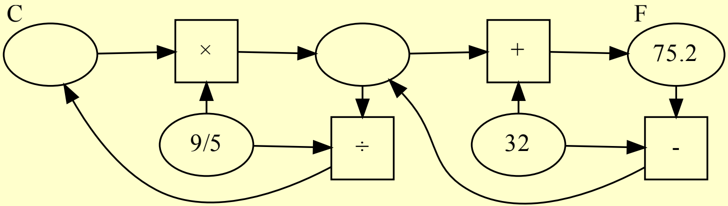
$$°F = °C \times \frac{9}{5} + 32$$

$$°C = (°F - 32) \div \frac{9}{5}$$



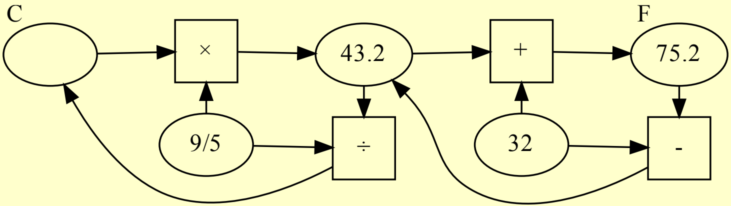
$$^{\circ}F = {^{\circ}C} \times \frac{9}{5} + 32$$

$$^{\circ}C = ({^{\circ}F} - 32) \div \frac{9}{5}$$



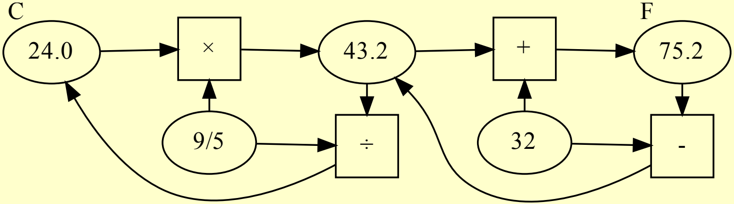
$$^\circ F = {^\circ C} \times \frac{9}{5} + 32$$

$$^\circ C = ({^\circ F} - 32) \div \frac{9}{5}$$



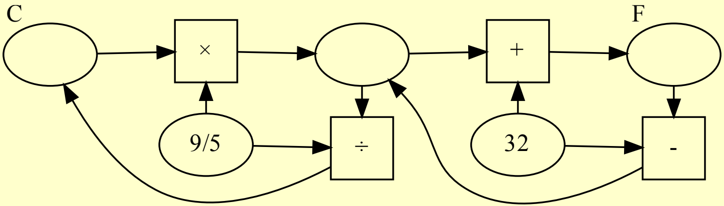
$$°F = °C \times \frac{9}{5} + 32$$

$$°C = (°F - 32) \div \frac{9}{5}$$



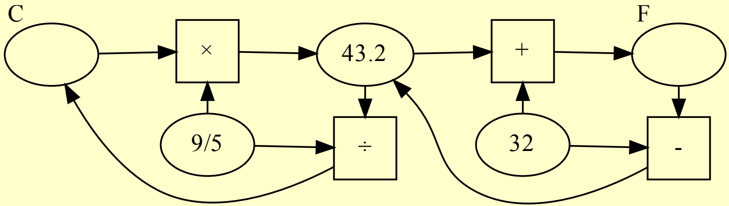
$$°F = °C \times \frac{9}{5} + 32$$

$$°C = (°F - 32) \div \frac{9}{5}$$



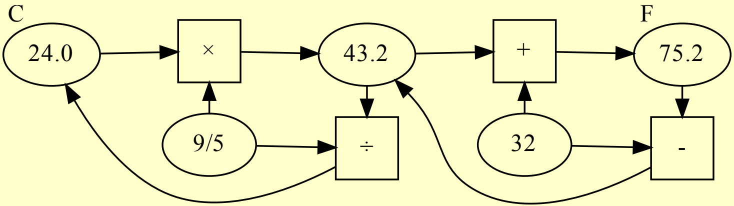
$$°F = °C \times \frac{9}{5} + 32$$

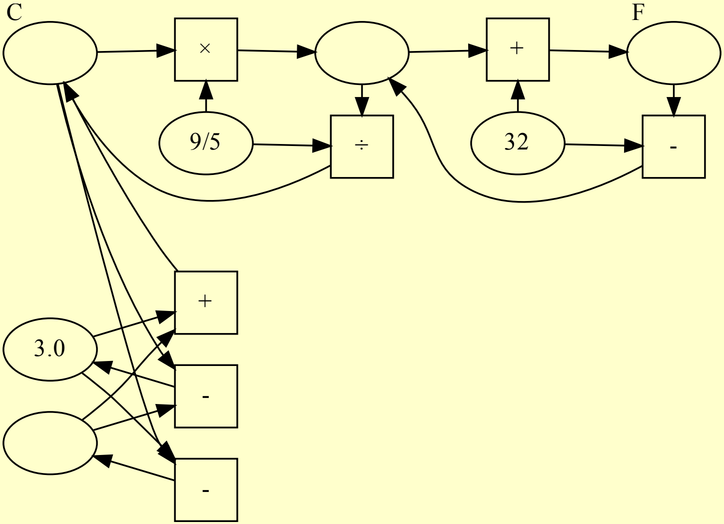
$$°C = (°F - 32) \div \frac{9}{5}$$



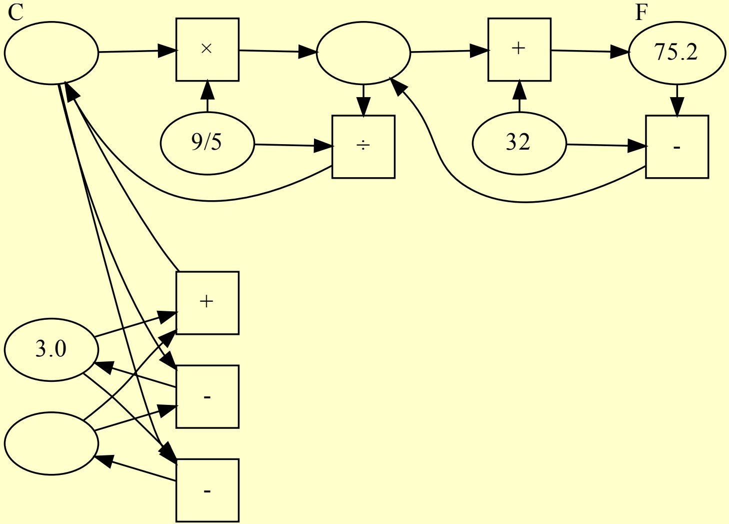
$$°F = °C \times \frac{9}{5} + 32$$

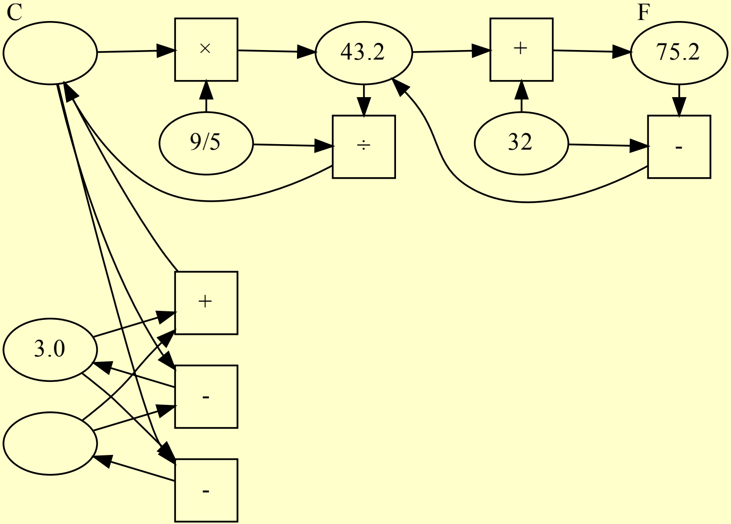
$$°C = (°F - 32) \div \frac{9}{5}$$



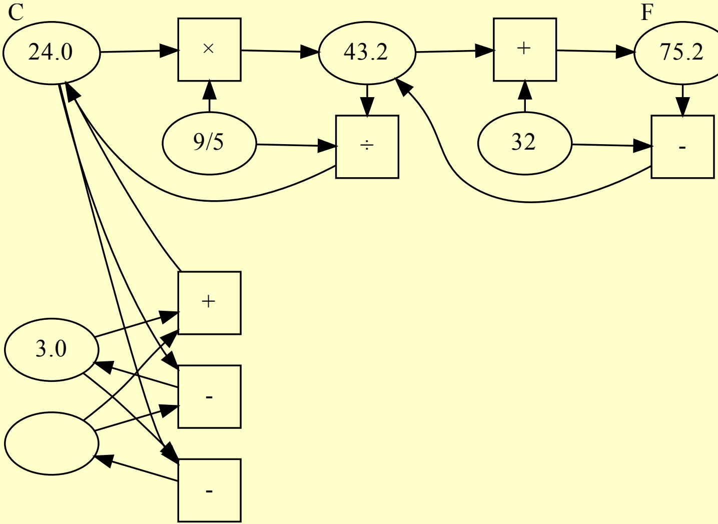


C
×
+
F
9/5
÷
32
-
+
3.0
-
-

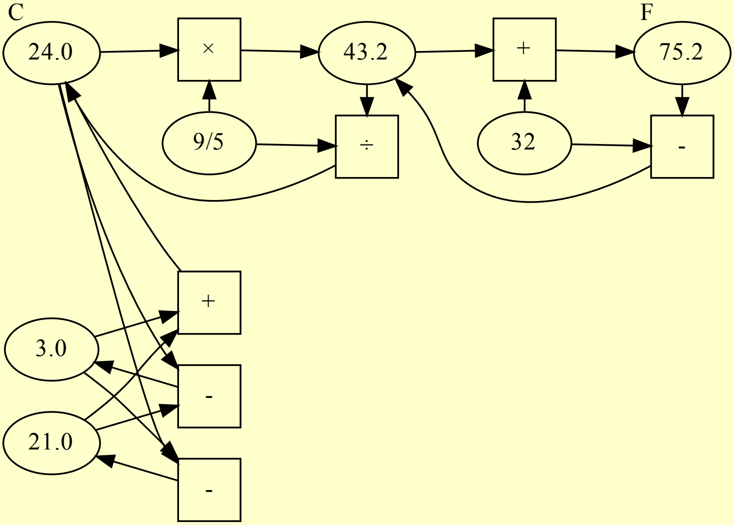

C
×
+
F
75.2
9/5
÷
32
-
+
3.0
-
-



C
×
43.2
+
F
75.2
9/5
÷
32
-
+
3.0
-
-



C
24.0
×
43.2
+
F
75.2
9/5
÷
32
-
+
3.0
-
-



C
24.0
×
43.2
+
F
75.2
9/5
÷
32
-
+
3.0
-
21.0
-

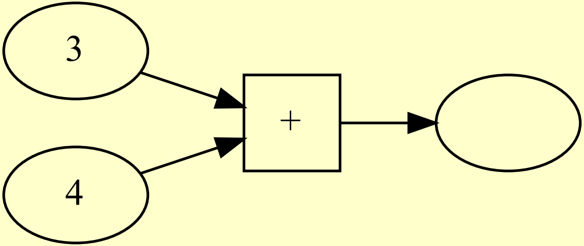
We can combine networks into larger networks!

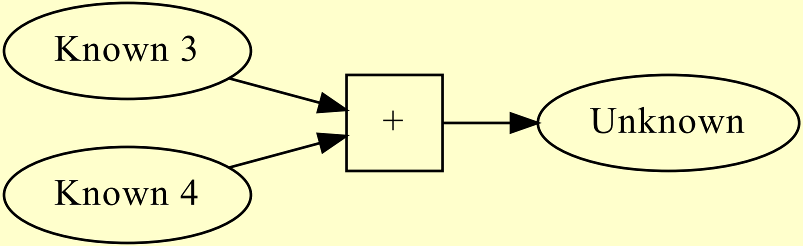
?

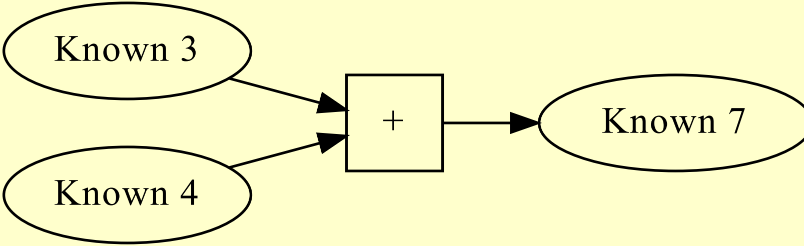
Cells *accumulate information* about a value

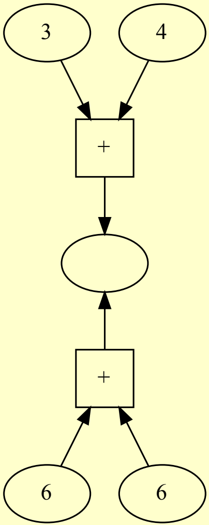
```
data Perhaps a = Unknown | Known a | Contradiction
```

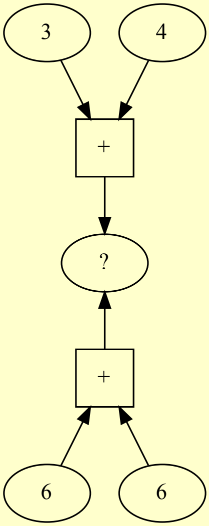
```haskell
data Perhaps a = Unknown | Known a | Contradiction

instance Eq a => Monoid (Perhaps a) where
  mempty = Unknown
  mappend Unknown x             = x
  mappend x       Unknown       = x
  mappend Contradiction _       = Contradiction
  mappend _     Contradiction = Contradiction
  mappend (Known a) (Known b) =
    if a == b
      then Known a
      else Contradiction
```

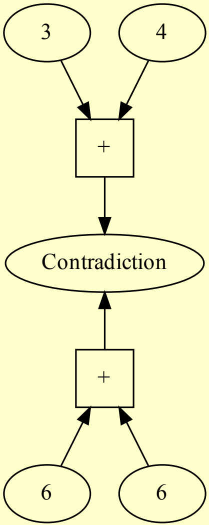

3
4
+

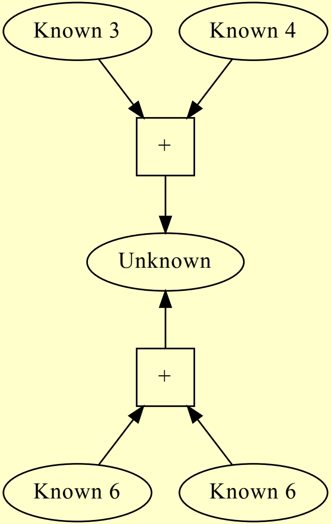

Known 3
Known 4
+
Unknown

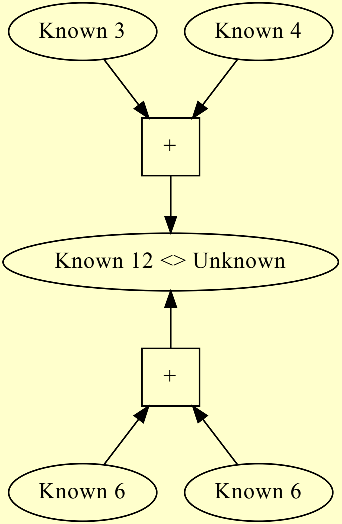

Known 3
Known 4
+
Known 7

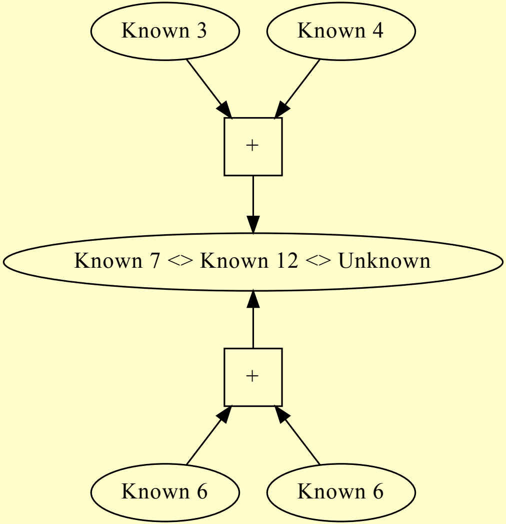

3
4
+
+
6
6

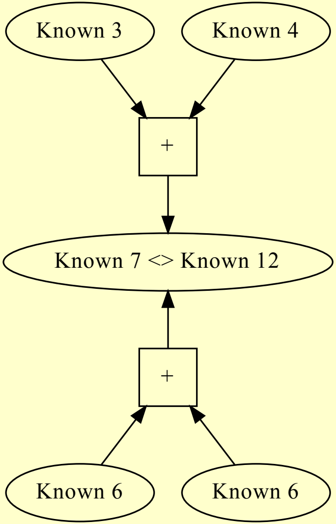

3
4
+
?
+
6
6

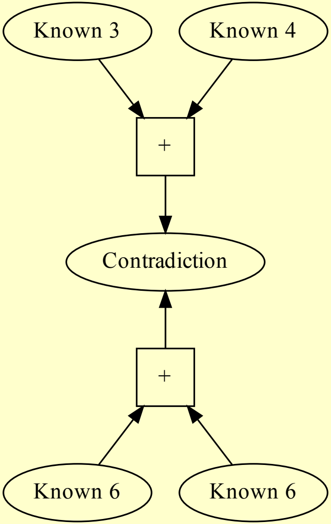

3
4
+
Contradiction
+
6
6


Known 3
Known 4
+
Unknown
+
Known 6
Known 6

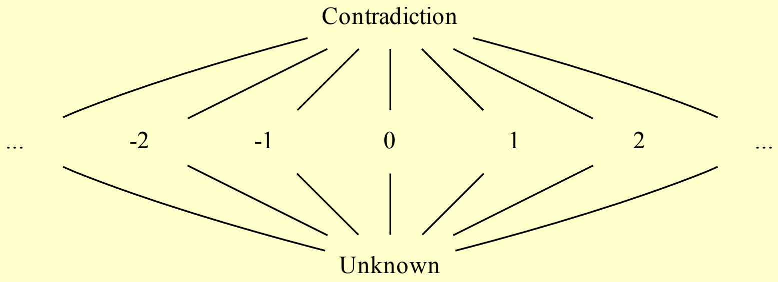

Known 3
Known 4
+
Known 12 <> Unknown
+
Known 6
Known 6

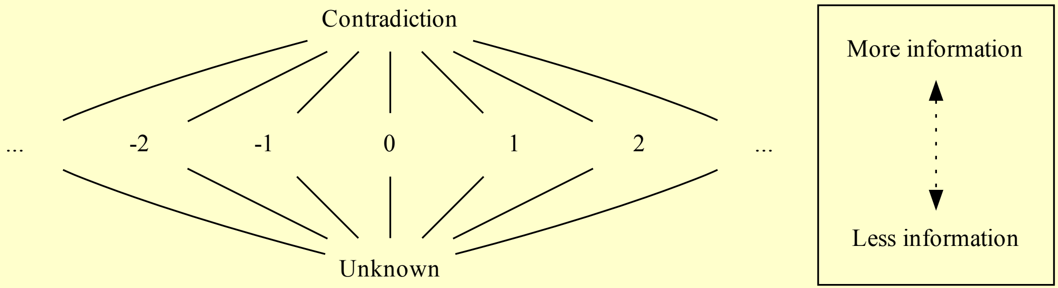

Known 3
Known 4
+
Known 7 <> Known 12 <> Unknown
+
Known 6
Known 6


Known 3
Known 4
+
Known 7 <> Known 12
+
Known 6
Known 6



Known 3
Known 4
+
Contradiction
+
Known 6
Known 6


Contradiction
...
-2
-1
0
1
2
...
Unknown


Contradiction
...
-2
-1
0
1
2
...
Unknown
More information
Less information

3			2
	4	1	
	3	2	
4			1

3			2
	4	1	
	3	2	
4			1

3			2
	4	1	
	3	2	
4			1

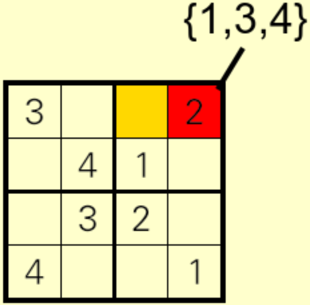
3			2
	4	1	
	3	2	
4			1

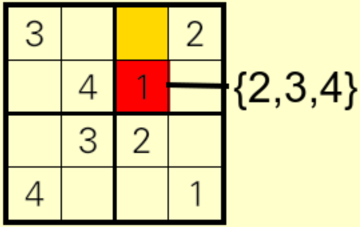
3			2
	4	1	
	3	2	
4			1

3			2
	4	1	
	3	2	
4			1

{1,2,3,4}

3			2
	4	1	
	3	2	
4			1


{1,3,4}
3 2
4 1
3 2
4 1


{2,3,4}

{1,2,4}

3			2
	4	1	
	3	2	
4			1

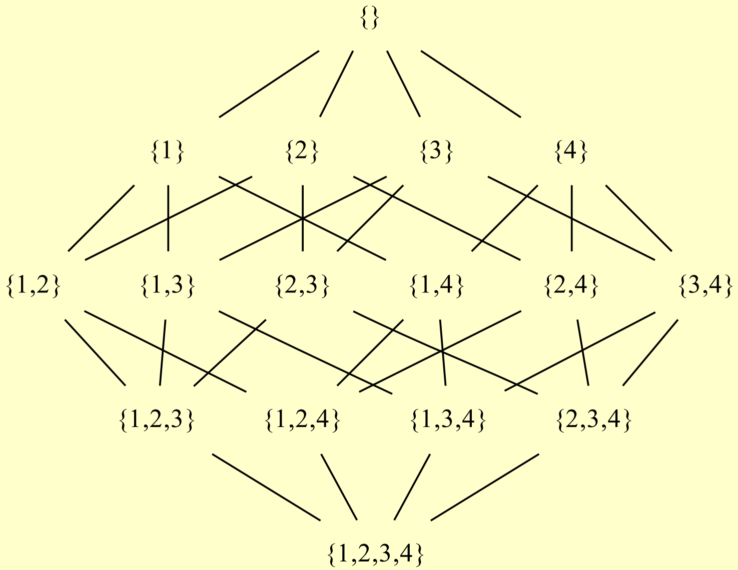
$$\{2,3,4\} \cap \{1,3,4\} \cap \{1,2,4\} \cap \{1,2,3\}$$

3			2
	4	1	
	3	2	
4			1

{4}

3			2
	4	1	
	3	2	
4			1

3		4	2
	4	1	
	3	2	
4			1

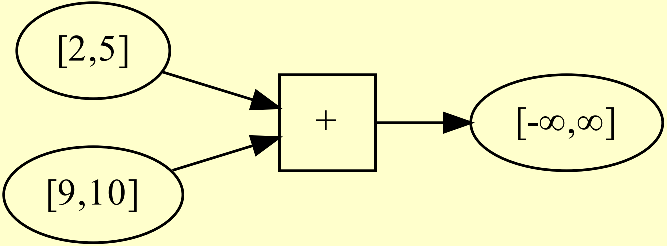

{}
{1} {2} {3} {4}
{1,2} {1,3} {2,3} {1,4} {2,4} {3,4}
{1,2,3} {1,2,4} {1,3,4} {2,3,4}
{1,2,3,4}

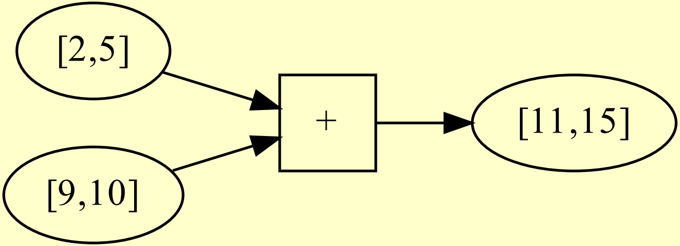
$$[1, 5]$$

$$[1, 5] \cup [2, 7] = [2, 5]$$

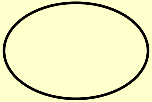
$$[1,5] \cup [2,7] = [2,5]$$

$$[2,5] + [9,10] = [11,15]$$

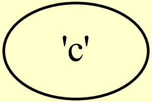

[2,5]
[9,10]
+
[-∞,∞]


[2,5]
[9,10]
+
[11,15]

What types are the values of the cells?



3


'c'

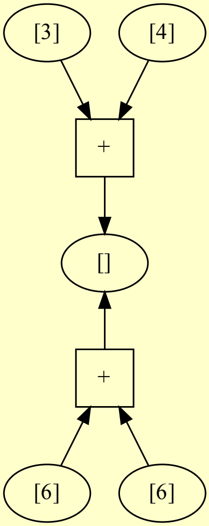
Contradiction

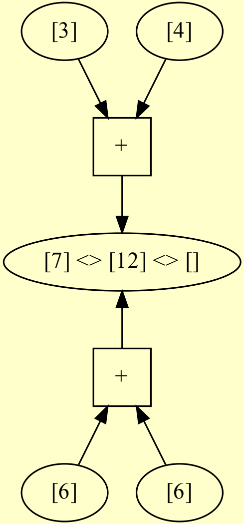
Is this the only type propagator cells can contain?

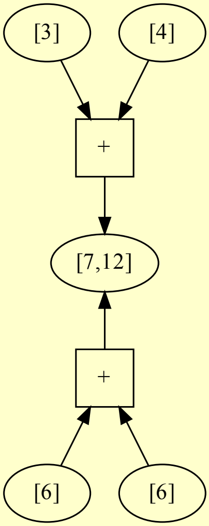
Will other monoids work?

Is this the only type propagator cells can contain?
Will other monoids work?

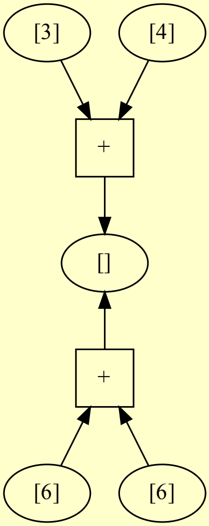
What about List?

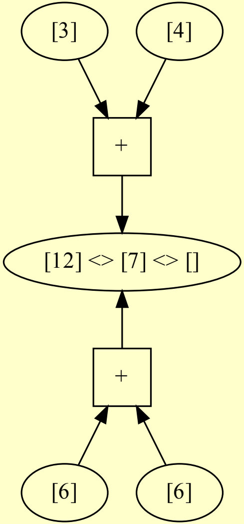

[3]
[4]
+
[]
+
[6]
[6]

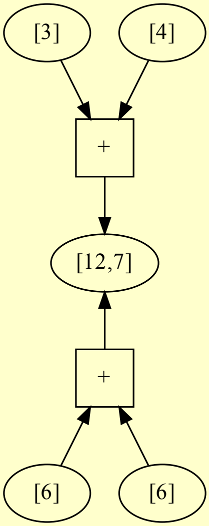

[3]
[4]
+
[7] <> [12] <> []
+
[6]
[6]

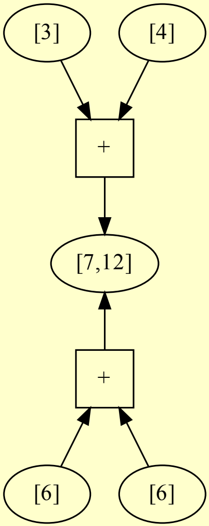

[3]
[4]
+
[7,12]
+
[6]
[6]

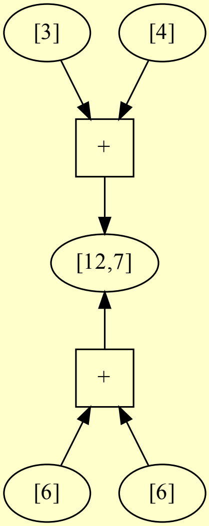
Looking good?


[3]
[4]
+
[]
+
[6]
[6]


[3]
[4]
+
[12] <> [7] <> []
+
[6]
[6]


[3]
[4]
+
[12,7]
+
[6]
[6]


[3]
[4]
+
[7,12]
+
[6]
[6]


[3]
[4]
+
[12,7]
+
[6]
[6]

We need commutativity!

$$x \oplus y = y \oplus x$$

We need commutativity!

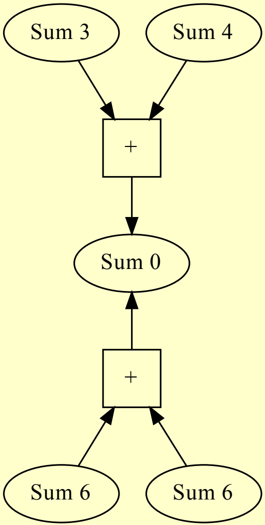
$$x \oplus y = y \oplus x$$

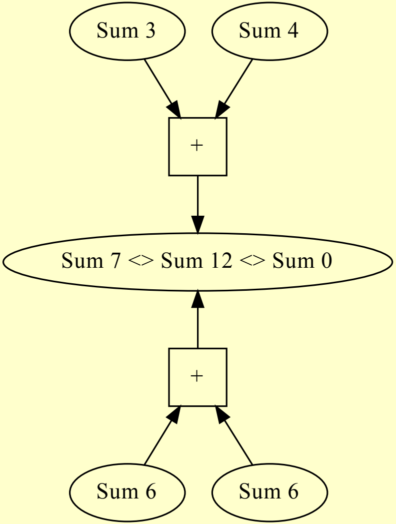
List append is not commutative!

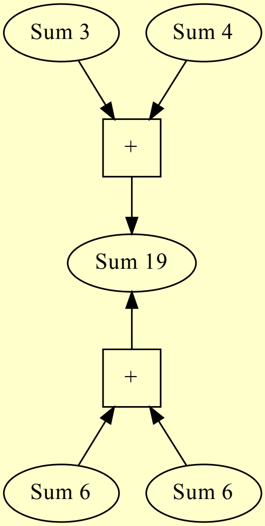
```
[1,2,3] <> [4,5,6] == [1,2,3,4,5,6]
[4,5,6] <> [1,2,3] == [4,5,6,1,2,3]
```

We need a commutative monoid
What about addition?

$$x + y = y + x$$


Sum 3
Sum 4
+
Sum 0
+
Sum 6
Sum 6


Sum 3
Sum 4
+
Sum 7 <> Sum 12 <> Sum 0
+
Sum 6
Sum 6


Sum 3
Sum 4
+
Sum 19
+
Sum 6
Sum 6

We need idempotence!

$$x \oplus x = x$$

We need an idempotent, commutative monoid.
This structure is called a *join-semilattice*

Associativity

$$(x \vee y) \vee z = x \vee (y \vee z)$$

Commutativity

$$x \vee y = y \vee x$$

Idempotence

$$x \vee x = x$$

Alexey Radul's work on propagators:

- Art of the Propagator
  `http://web.mit.edu/~axch/www/art.pdf`
- Propagation Networks: A Flexible and Expressive Substrate for Computation
  `http://web.mit.edu/~axch/www/phd-thesis.pdf`

Lindsey Kuper's work on LVars is closely related, and works today:

- Lattice-Based Data Structures for Deterministic Parallel and Distributed Programming
  `https://www.cs.indiana.edu/~lkuper/papers/lindsey-kuper-dissertation.pdf`
- lvish library
  `https://hackage.haskell.org/package/lvish`

Edward Kmett has worked on:

- Making propagators go fast
- Scheduling strategies and garbage collection
- Relaxing requirements (Eg. not requiring a full join-semilattice, admitting non-monotone functions)

Ed's stuff:

- `http://github.com/ekmett/propagators`
- `http://github.com/ekmett/concurrent`
- Lambda Jam talk (Easy mode): `https://www.youtube.com/watch?v=acZkF6Q2XKs`
- Boston Haskell talk (Hard mode): `https://www.youtube.com/watch?v=DyPzPeOPgUE`

In conclusion, propagator networks:

- Admit any Haskell function you can write today . . .
- . . . and more functions!
- compute bidirectionally
- give us constraint solving and search
- parallelise and distribute

Thanks for listening!