

Propagators: An Introduction

George Wilson

Data61/CSIRO

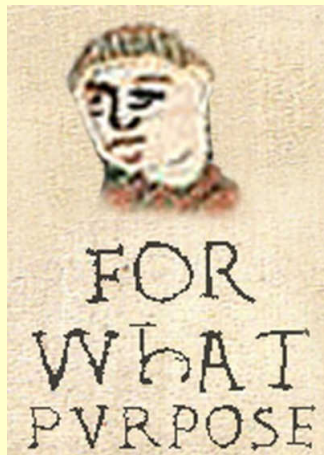
george.wilson@data61.csiro.au

November 13, 2017





What?



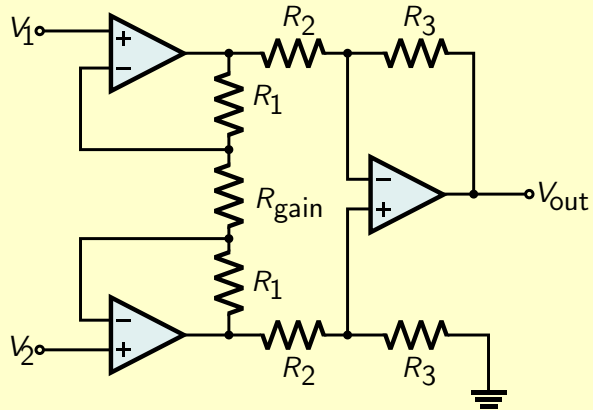
Why?

Beginnings as early as the 1970's at MIT

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

More recently:

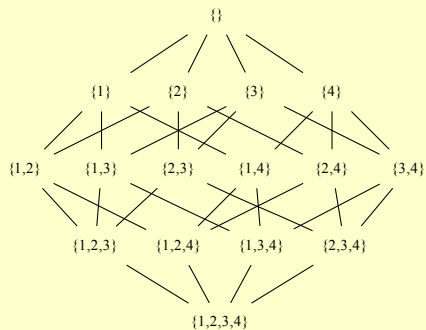
- Alexey Radul



```
(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*

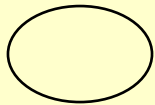
The *propagator model* is a model of computation
We model computations as *propagator networks*

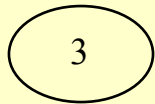
Propagator networks:

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

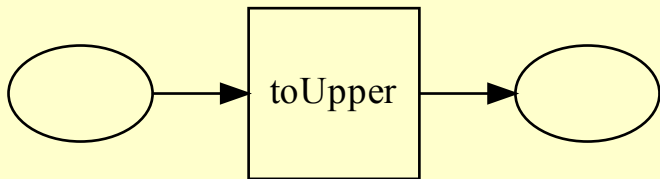
A propagator network comprises

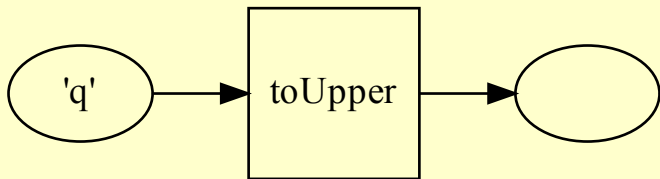
- cells
- propagators
- connections between cells and propagators

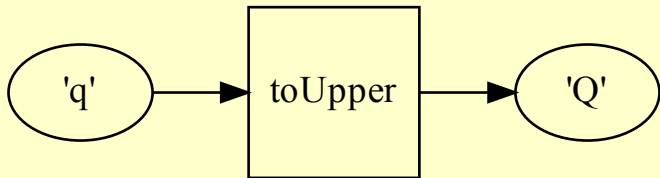


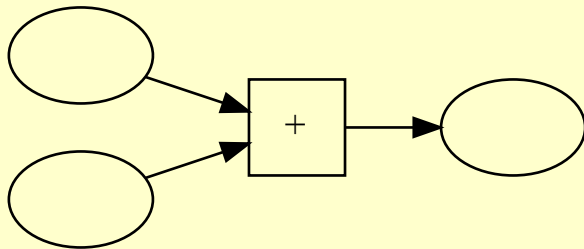


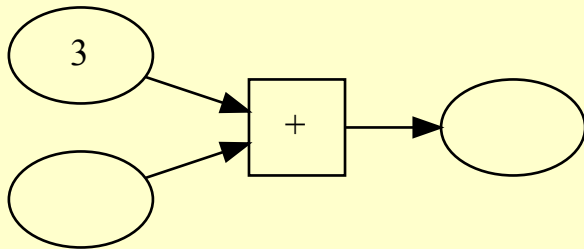
toUpper

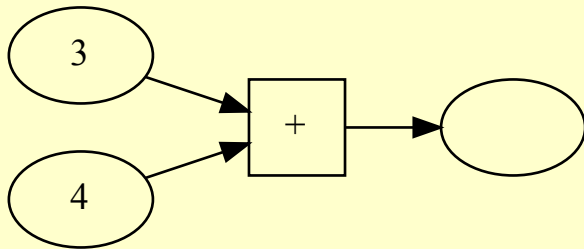


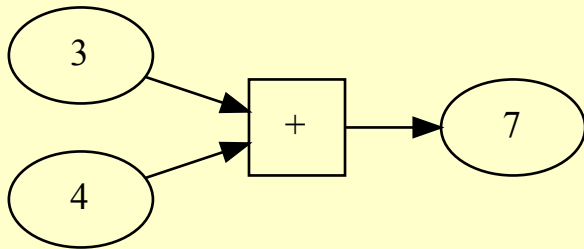












$$z \leftarrow x + y$$

$$z = x + y$$

$$7 = x + 4$$

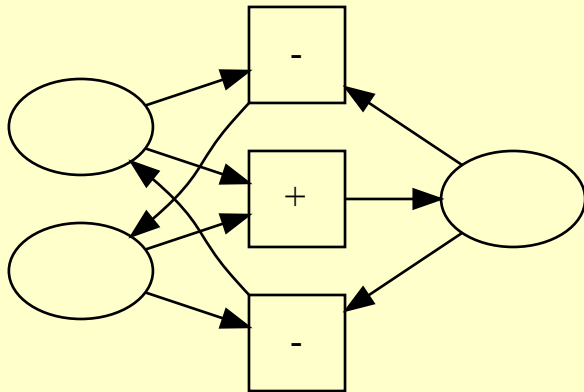
$$7 = 3 + 4$$

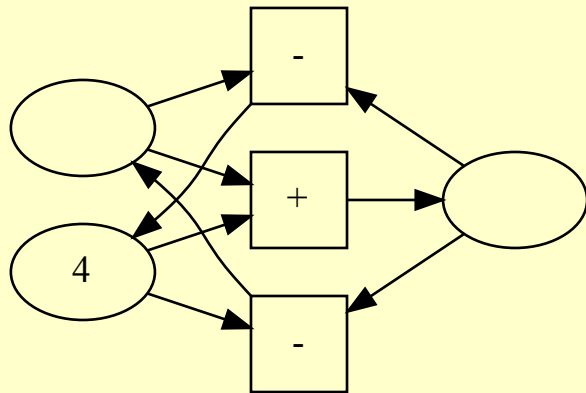
$$z = x + y$$

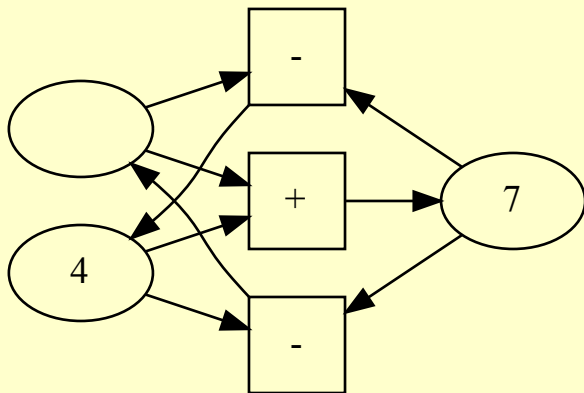
$$z \leftarrow x + y$$

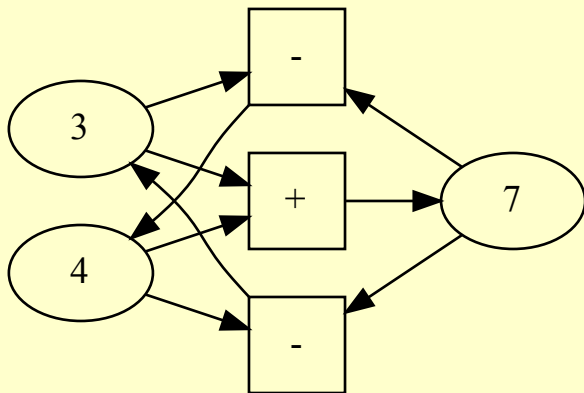
$$x \leftarrow z - y$$

$$y \leftarrow z - x$$



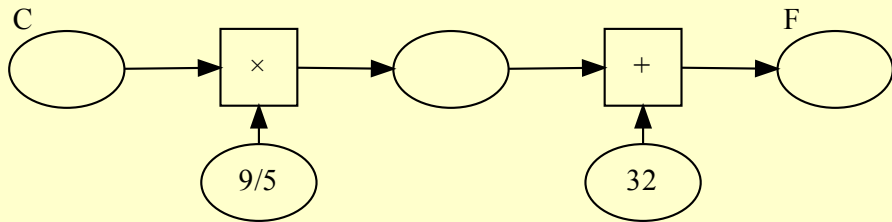




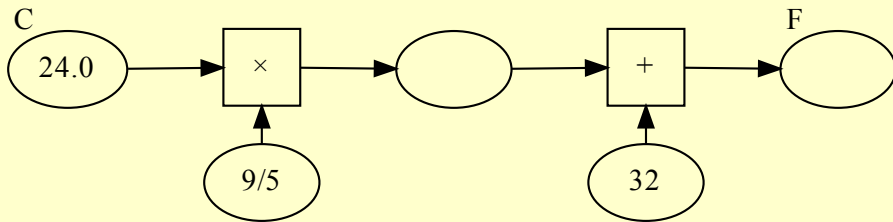


Propagators let us express multi-directional relationships!

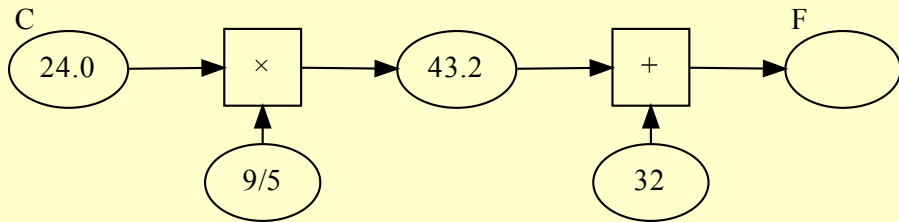
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$



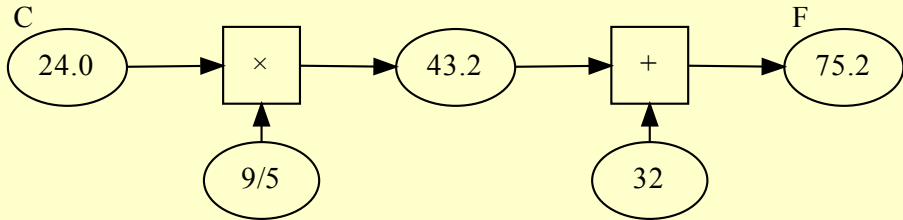
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$



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

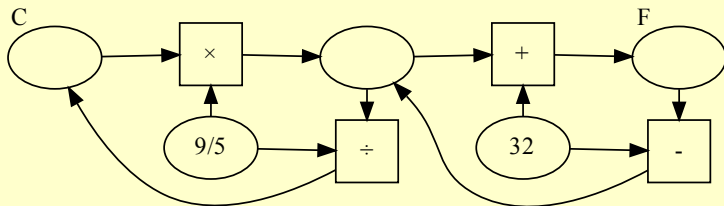


$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$



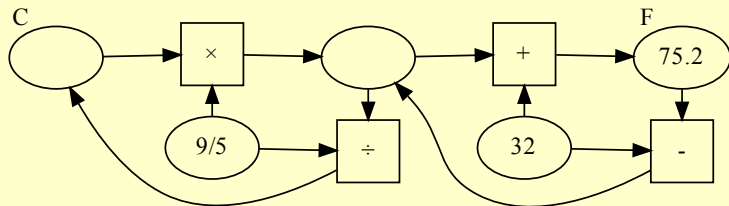
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$

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



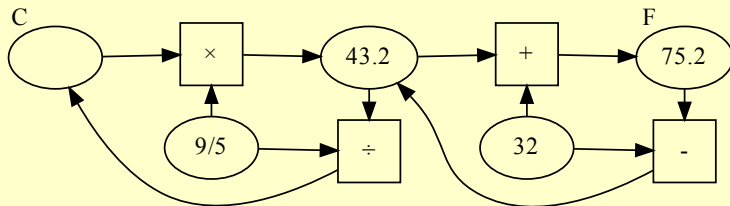
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$

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



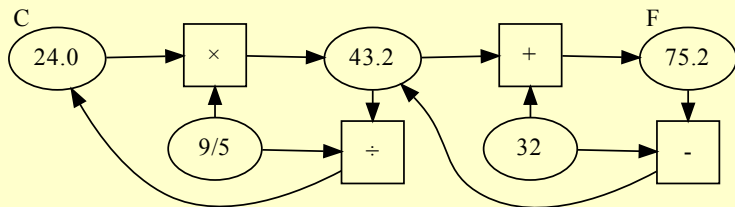
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$

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



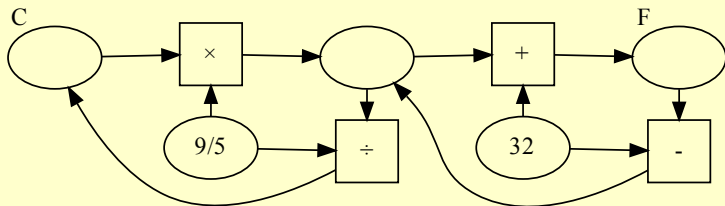
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$

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



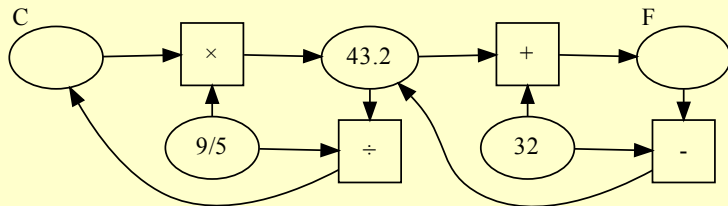
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$

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



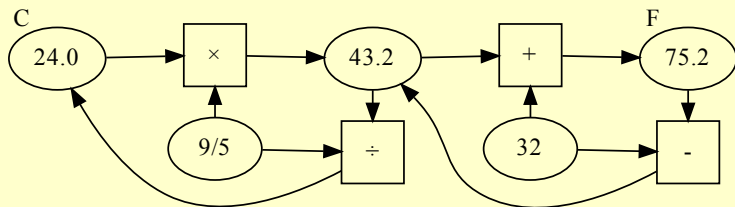
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$

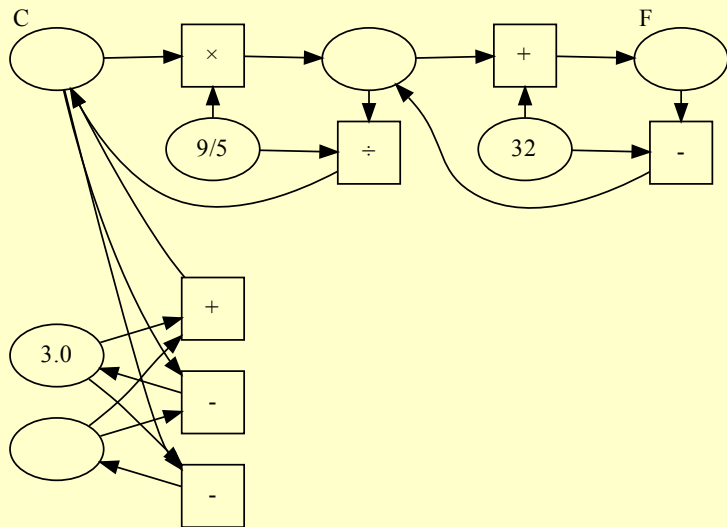
$$^{\circ}C = (^{\circ}F - 32) \div \frac{9}{5}$$

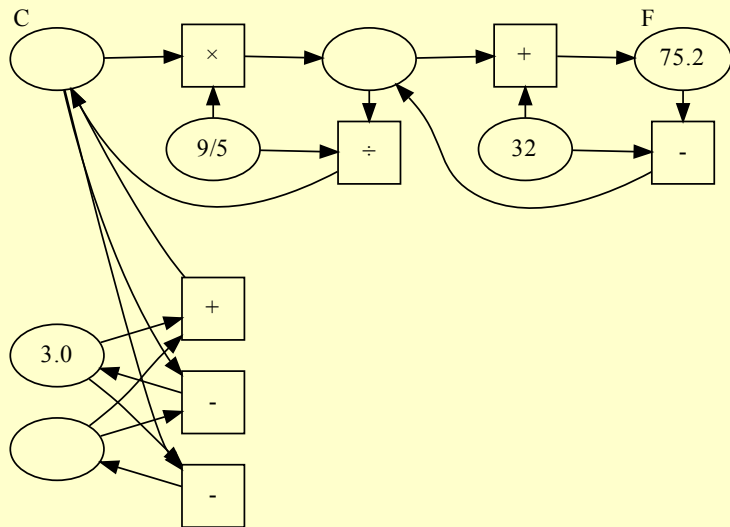


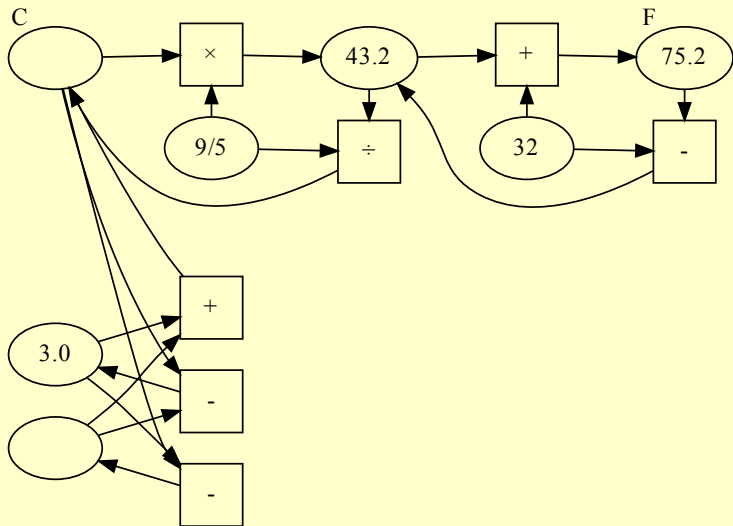
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$

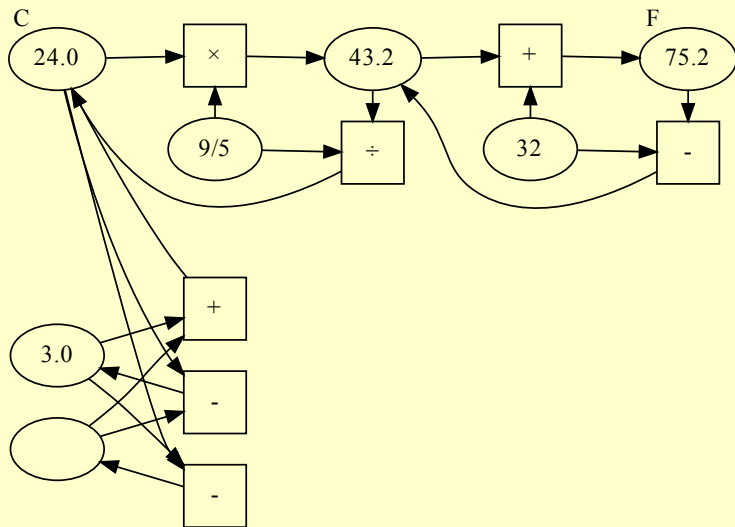
$$^{\circ}C = (^{\circ}F - 32) \div \frac{9}{5}$$

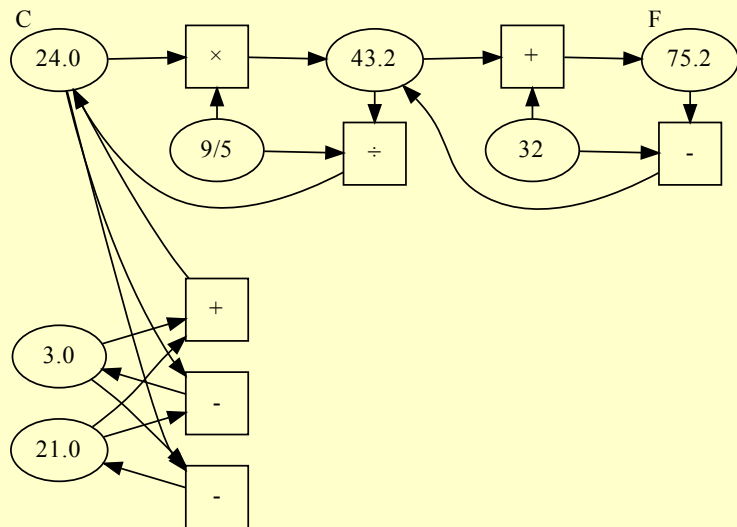












We can combine networks into larger networks!

?

Cells *accumulate information* about a value

3			2
	4	1	
	3	2	
4			1

3			2
	4	1	
	3	2	
4			1

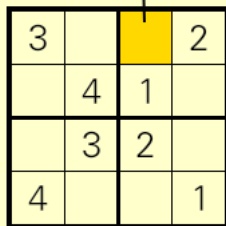
3			2
	4	1	
	3	2	
4			1

3			2
	4	1	
	3	2	
4			1

3			2
	4	1	
	3	2	
4			1

3			2
	4	1	
	3	2	
4			1

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



A 4x4 grid with a yellow cell at (1,3) and a pointer from the set {1,2,3,4} to it.

3			2
	4	1	
	3	2	
4			1

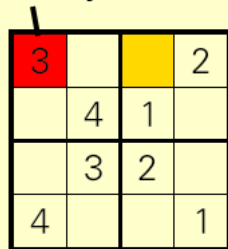
$\{1,3,4\}$

3			2
	4	1	
	3	2	
4			1

3			2
	4	1	
	3	2	
4			1

$\{2,3,4\}$

$\{1,2,4\}$



A 4x4 grid with a red cell containing the number 3 and a yellow cell. An arrow points from the set $\{1,2,4\}$ to the red cell.

3			2
	4	1	
	3	2	
4			1

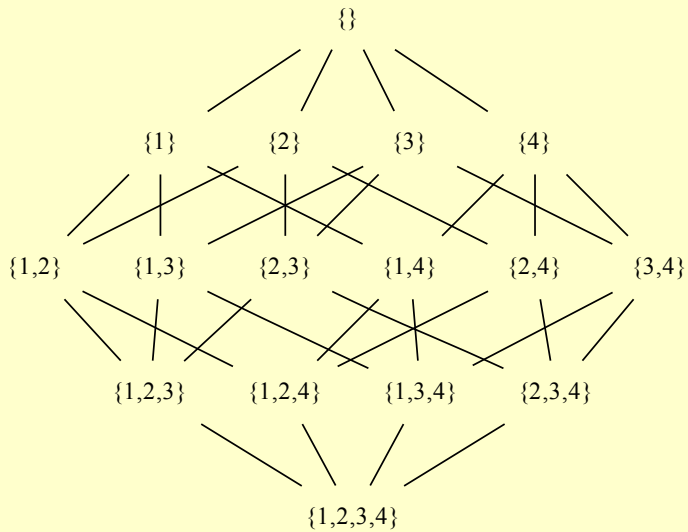
$$\{2,3,4\} \cap \{1,3,4\} \cap \\ \{1,2,4\} \cap \{1,2,3\}$$

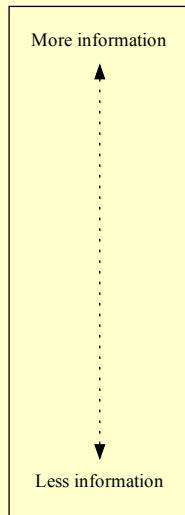
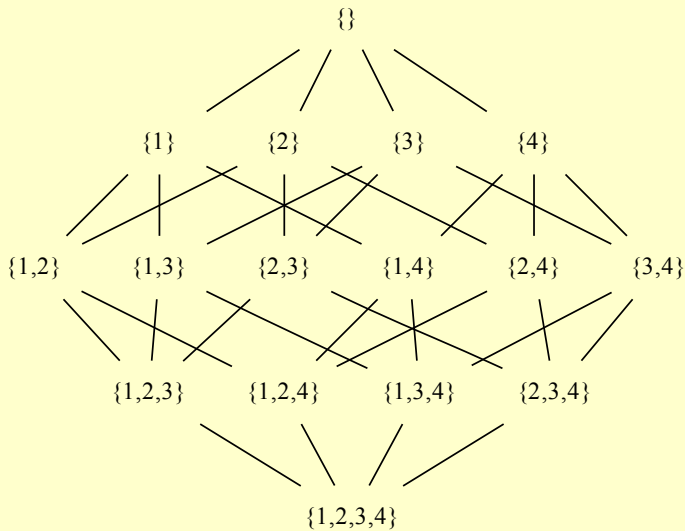
3			2
	4	1	
	3	2	
4			1

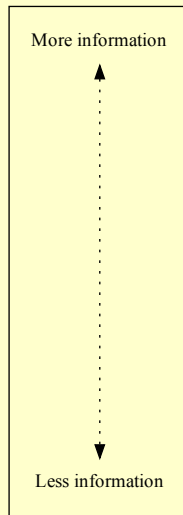
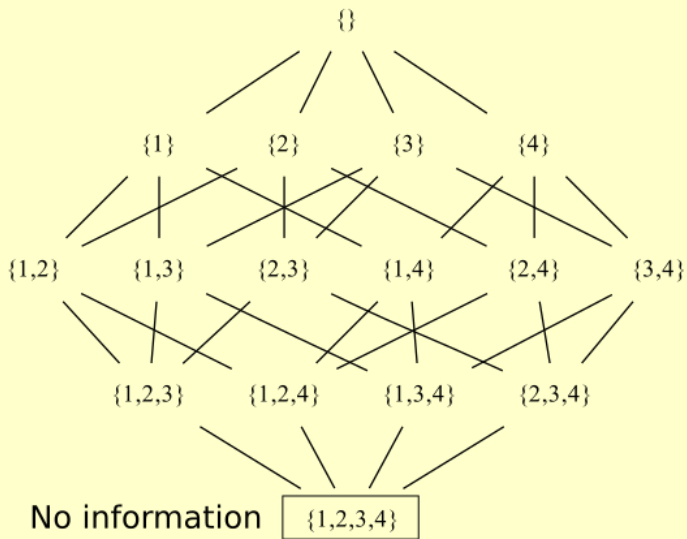
{4}

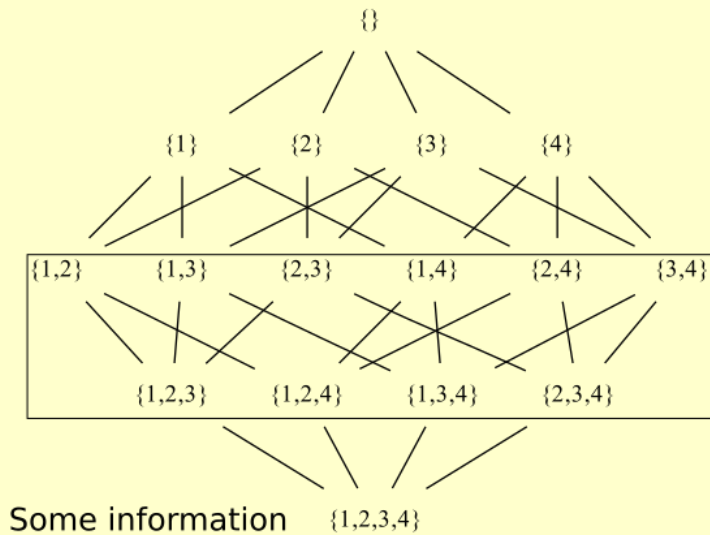
3			2
	4	1	
	3	2	
4			1

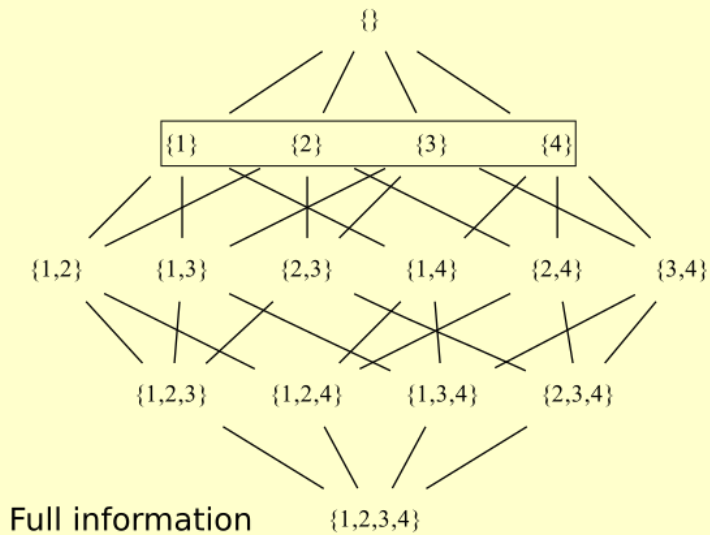
3		4	2
	4	1	
	3	2	
4			1

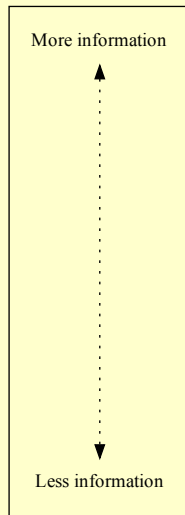
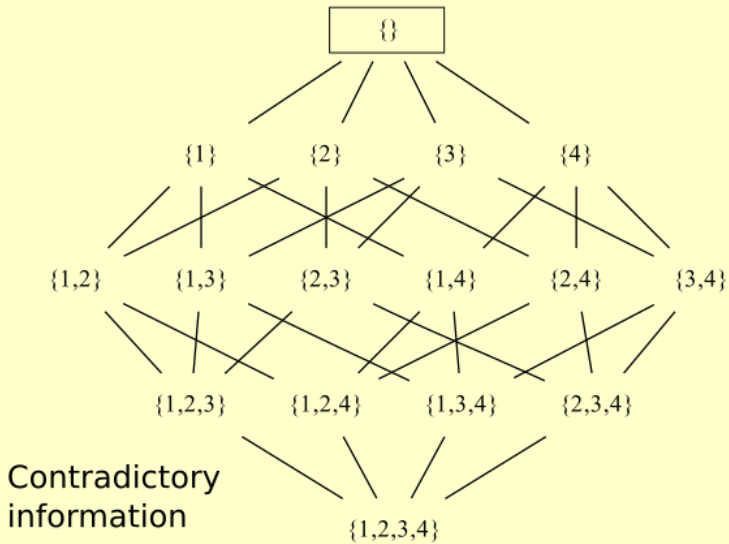












Cells accumulate information in a *bounded join-semilattice*

Cells accumulate information in a *bounded join-semilattice*

A bounded join-semilattice is:

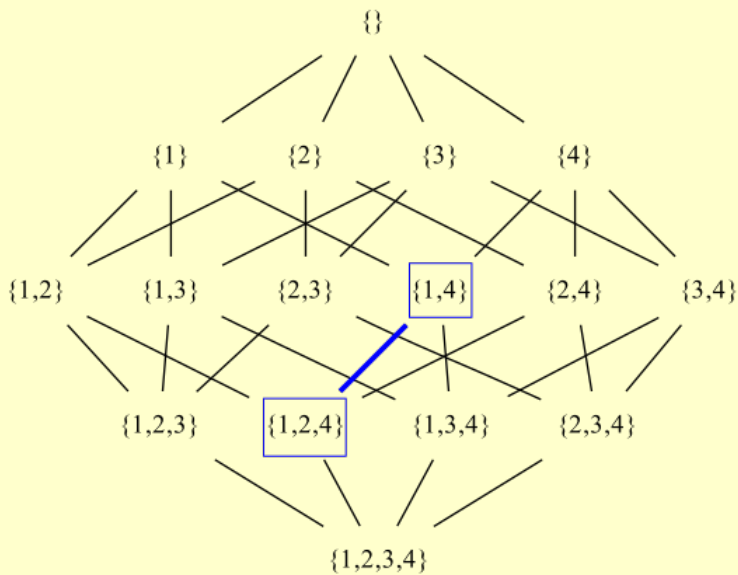
- A *partially ordered set*
- with a least element
- such that any set of elements has a *least upper bound*

Cells accumulate information in a *bounded join-semilattice*

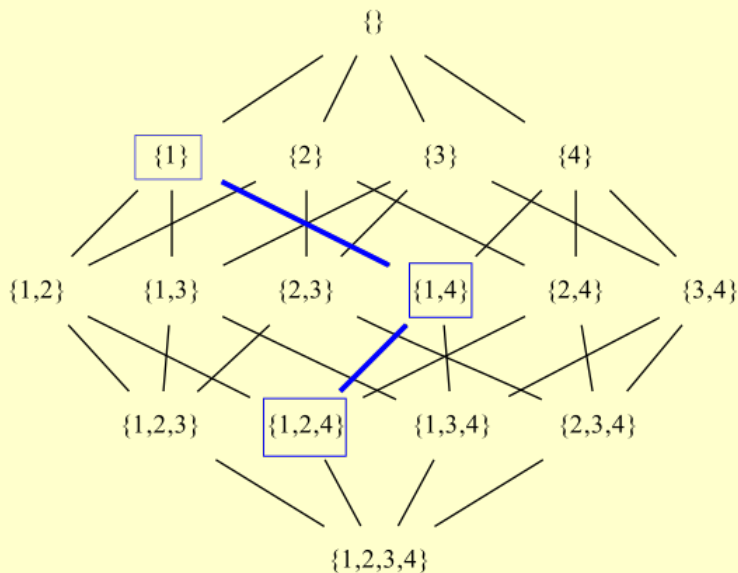
A bounded join-semilattice is:

- A *partially ordered set*
- with a least element
- such that any set of elements has a *least upper bound*

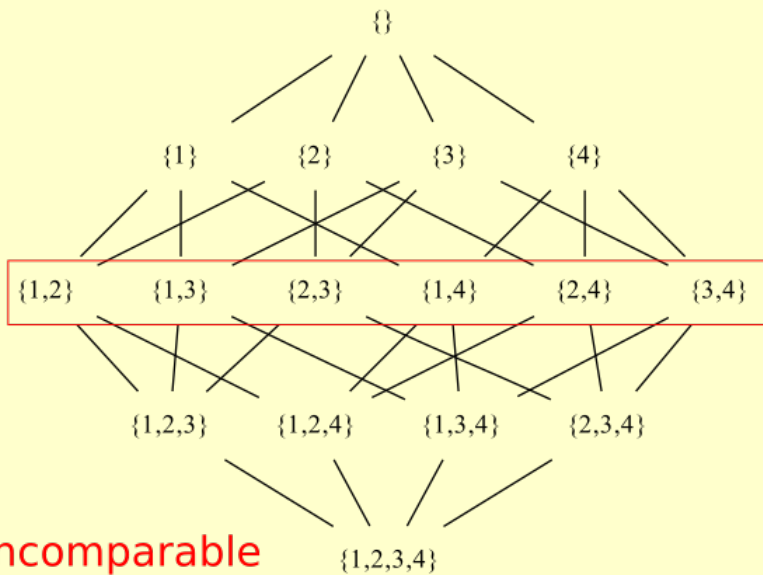
“Least upper bound” is denoted as \vee and is usually pronounced “join”

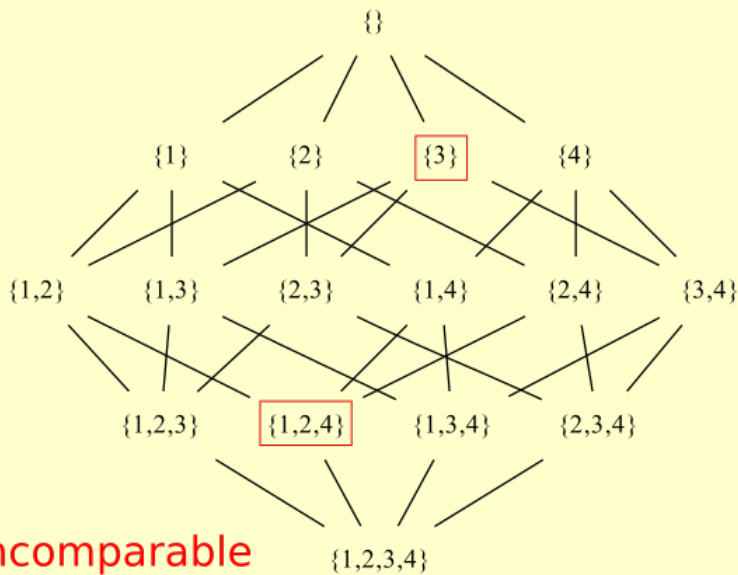


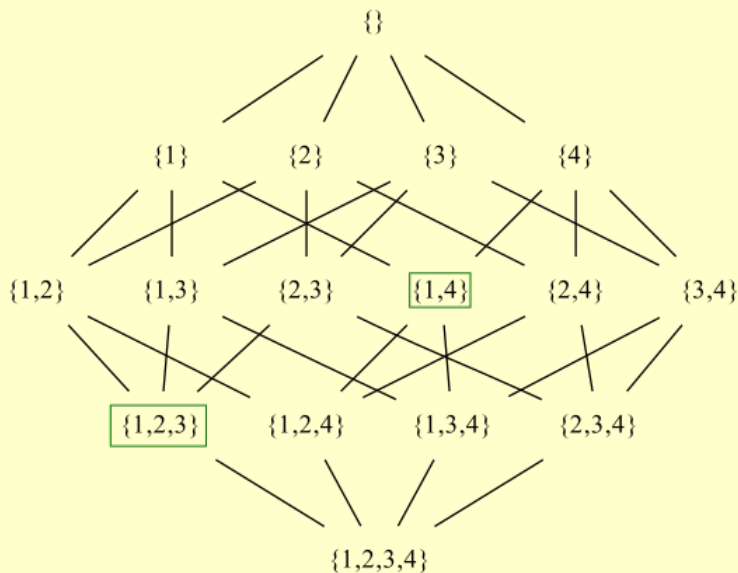
$\{1,2,4\} < \{1,4\}$



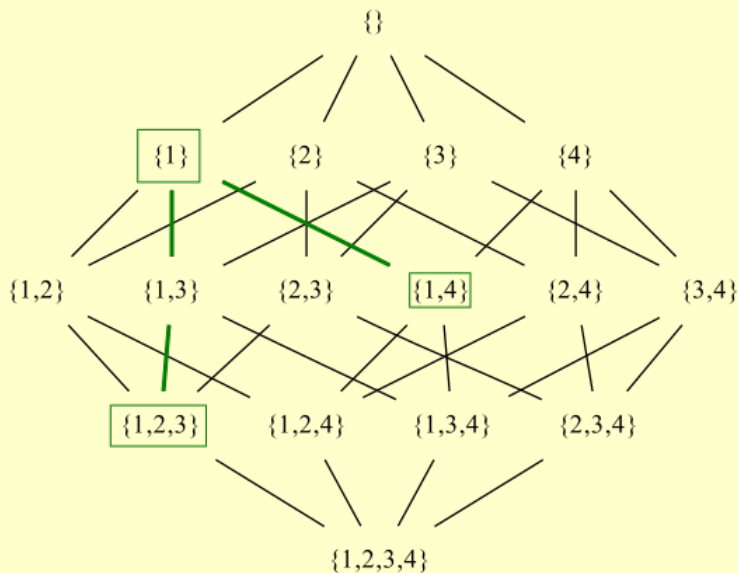
$$\{1,2,4\} < \{1,4\} < \{1\}$$







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



$$\{1,2,3\} \vee \{1,4\} = \{1\}$$

\vee has useful algebraic properties. It is:

- A monoid
- that's commutative
- and idempotent

Left identity

$$\epsilon \vee x = x$$

Right identity

$$x \vee \epsilon = x$$

Associativity

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

Commutative

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

Idempotent

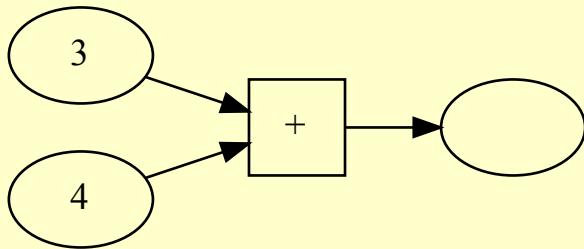
$$x \vee x = x$$

These laws and some other conditions make propagator networks deterministic, even in the face of parallelism and distribution

These laws and some other conditions make propagator networks deterministic, even in the face of parallelism and distribution

Bounded join-semilattices are already popular in the distributed systems world
See: Conflict Free Replicated Datatypes

?



data Perhaps $a = \text{Unknown}$ | $\text{Known } a$ | Contradiction

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

```
instance Eq a => BoundedJoinSemiLattice (Perhaps a) where
```

```
    bottom = Unknown
```

```
    (\\) Unknown x           = x
```

```
    (\\) x           Unknown = x
```

```
    (\\) Contradiction _      = Contradiction
```

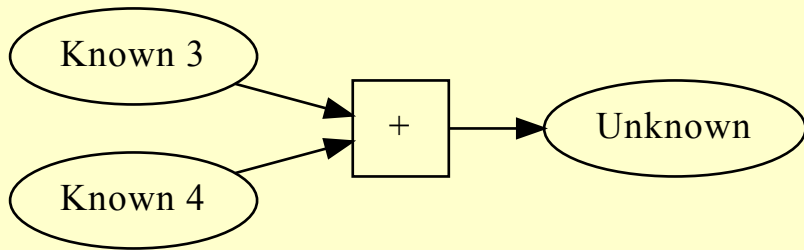
```
    (\\) _      Contradiction = Contradiction
```

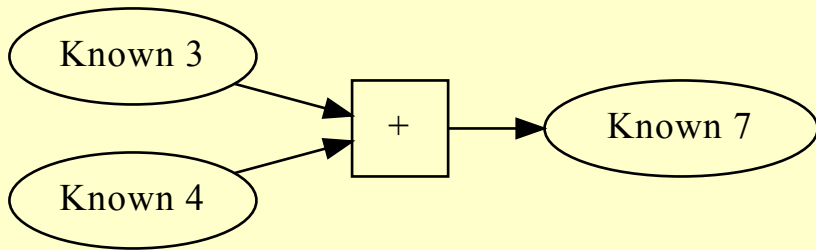
```
    (\\) (Known a) (Known b) =
```

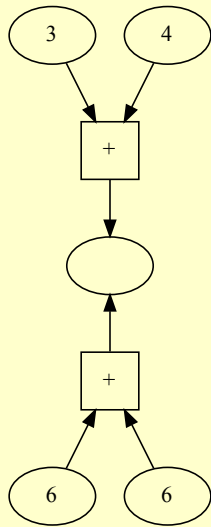
```
        if a == b
```

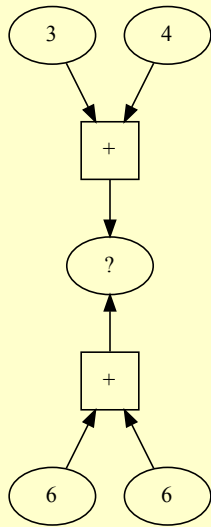
```
            then Known a
```

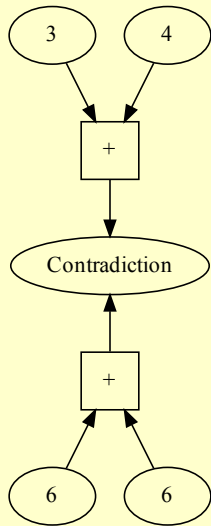
```
            else Contradiction
```

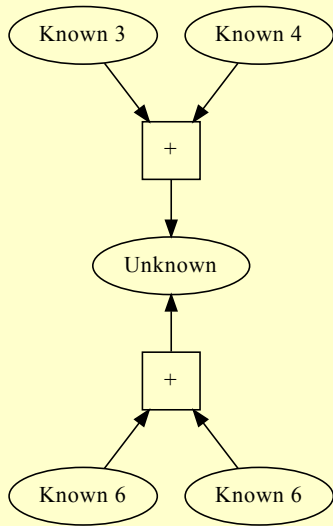


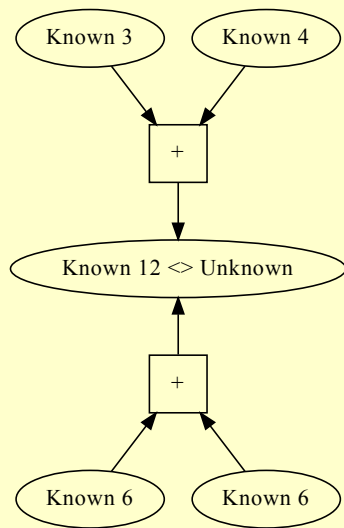


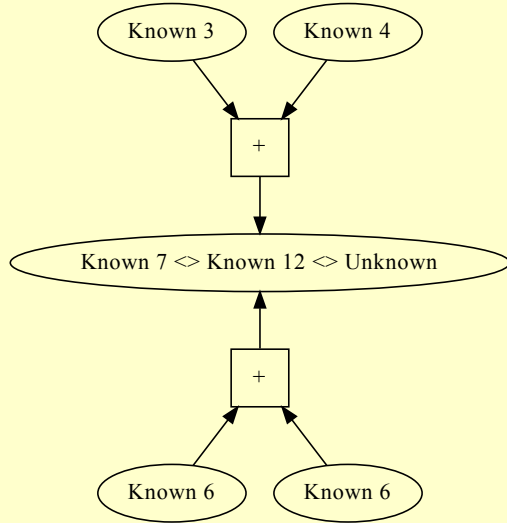


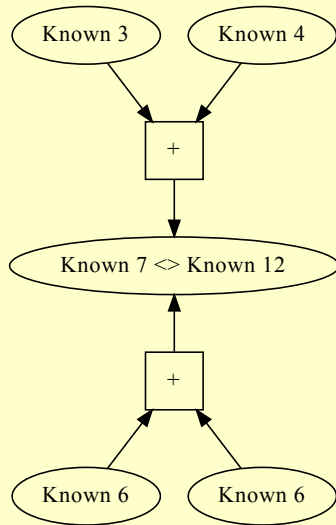


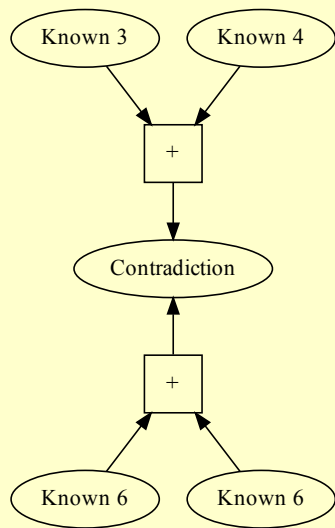


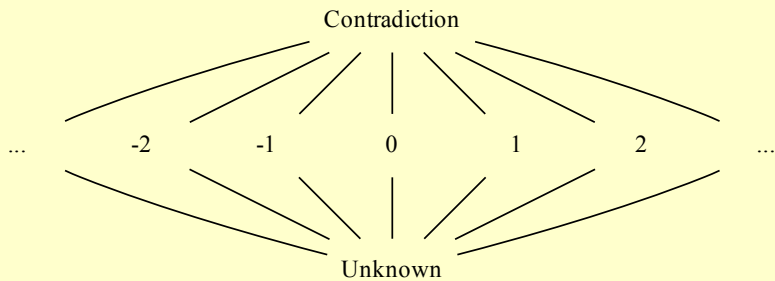


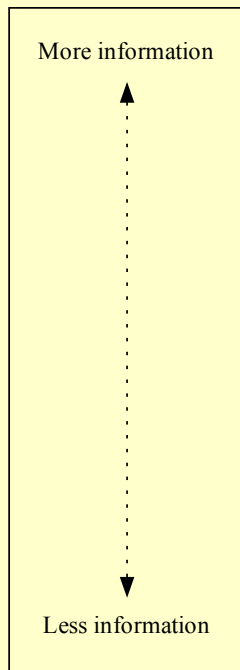
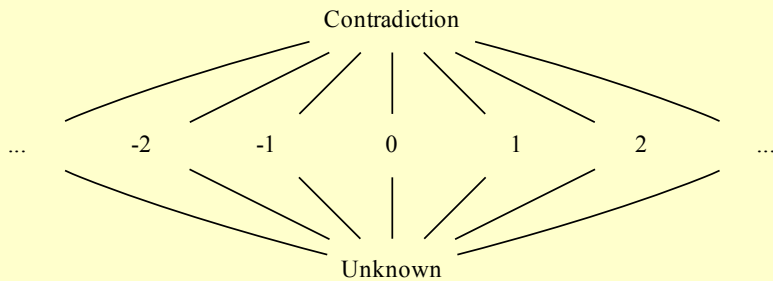










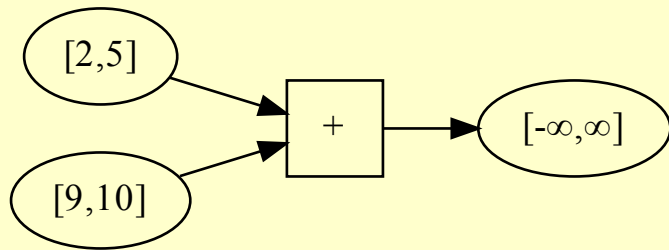


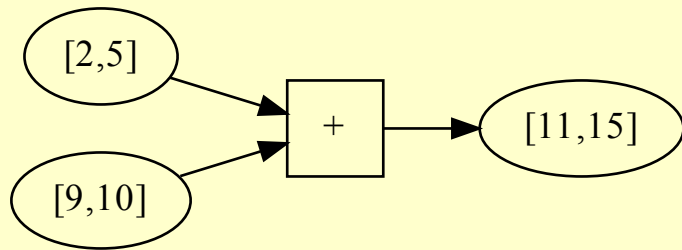
$$[1, 5]$$

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

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

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





(more interval stuff)

(Talk about truth-management system machinery)

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!