

# An Intuition for Propagators

George Wilson

CSIRO's Data61

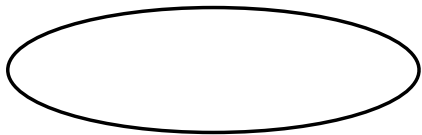
[george.wilson@data61.csiro.au](mailto:george.wilson@data61.csiro.au)

2nd September 2019



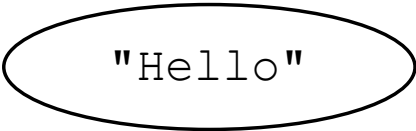
1970s, MIT

a model of computation for **highly parallel** machines



do

c <- cell



"Hello"

do

c <- cell

write c "Hello"



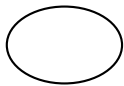
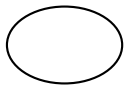
"Compose"

do

c <- cell

write c "Hello"

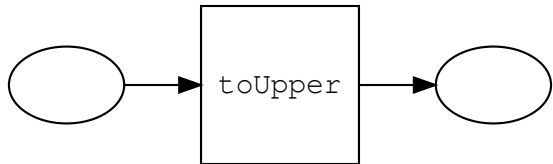
write c "Compose"



do

input <- cell

output <- cell



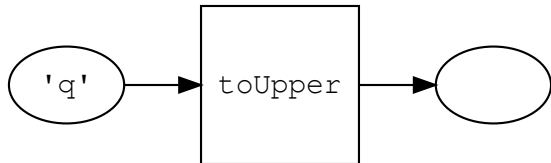
do

input <- cell

output <- cell

lift toUpper input output





do

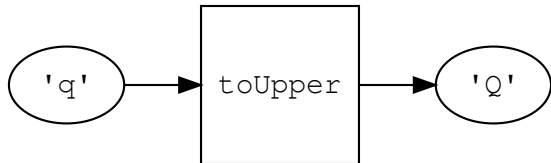
```
input  <- cell
```

```
output <- cell
```

```
lift toUpper input output
```

```
-- run the network
```

```
write input 'q'
```



do

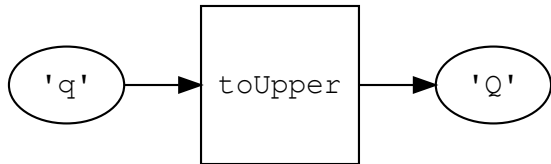
```
input  <- cell
```

```
output <- cell
```

```
lift toUpper input output
```

```
-- run the network
```

```
write input 'q'
```



do

```
input  <- cell
```

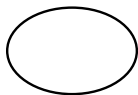
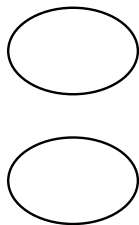
```
output <- cell
```

```
lift toUpper input output
```

```
-- run the network
```

```
write input 'q'
```

```
content output    -- Just 'Q'
```

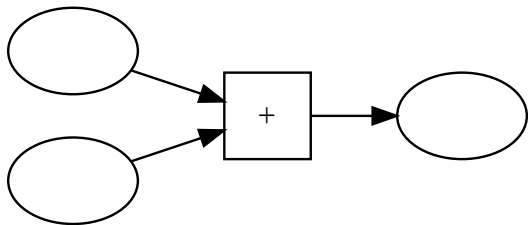


do

inL <- cell

inR <- cell

out <- cell



**do**

inL <- cell

inR <- cell

out <- cell

adder inL inR out

**where**

adder l r o = **do**

lift2 (+) l r o

$$z = x + y$$

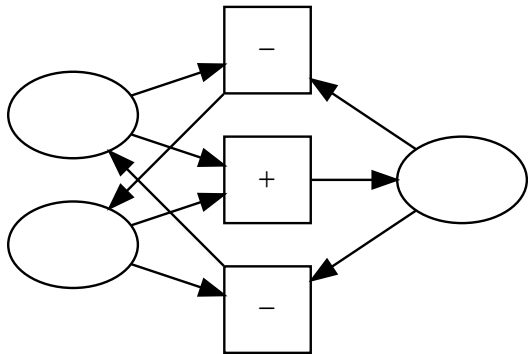
$$z \leftarrow x + y$$

$$z \leftarrow x + y$$

$$x \leftarrow z - y$$

$$y \leftarrow z - x$$





**do**

inL  $\leftarrow$  cell

inR  $\leftarrow$  cell

out  $\leftarrow$  cell

adder inL inR out

**where**

adder l r o = **do**

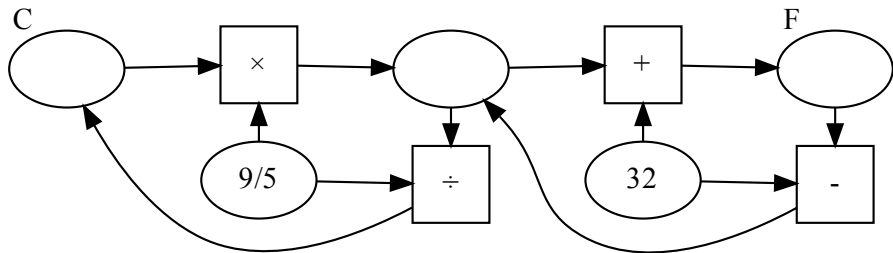
lift2 (+) l r o

lift2 (-) o l r

lift2 (-) o r l

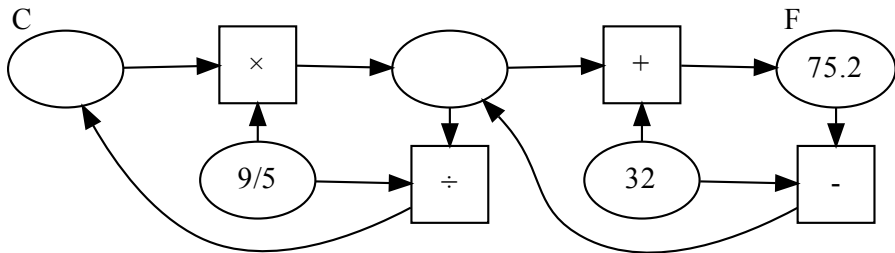
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$

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



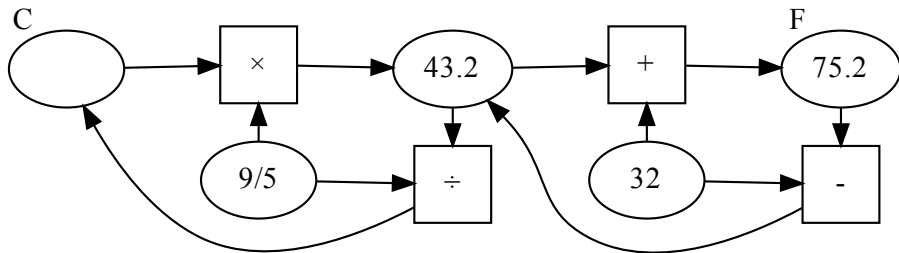
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$

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



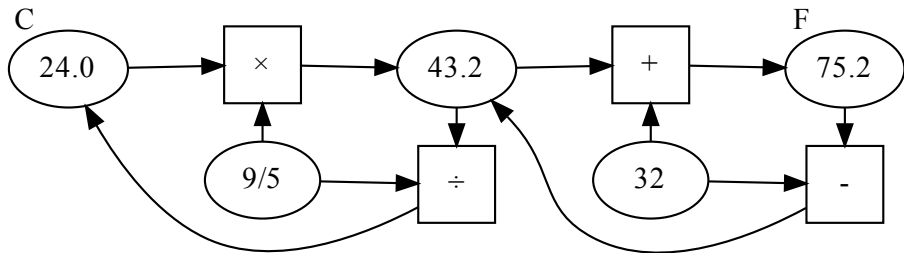
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$

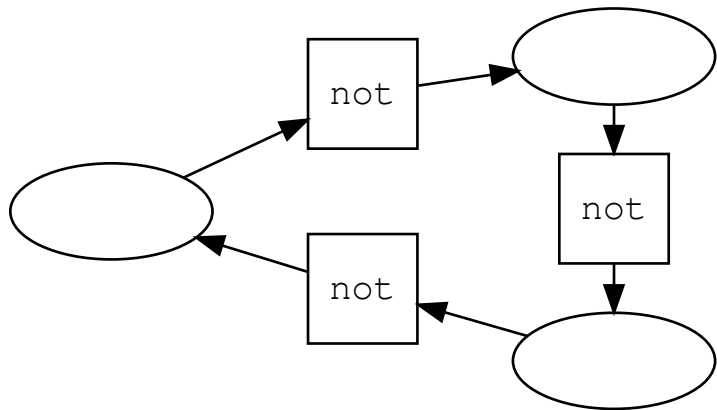
$$^{\circ}C = (^{\circ}F - 32) \div \frac{9}{5}$$

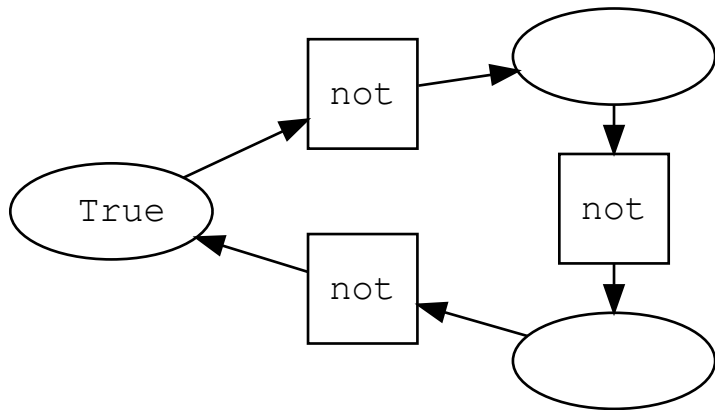


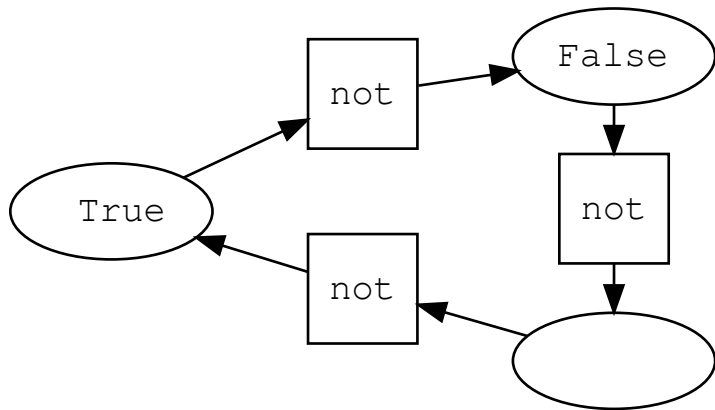
$$^{\circ}F = ^{\circ}C \times \frac{9}{5} + 32$$

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

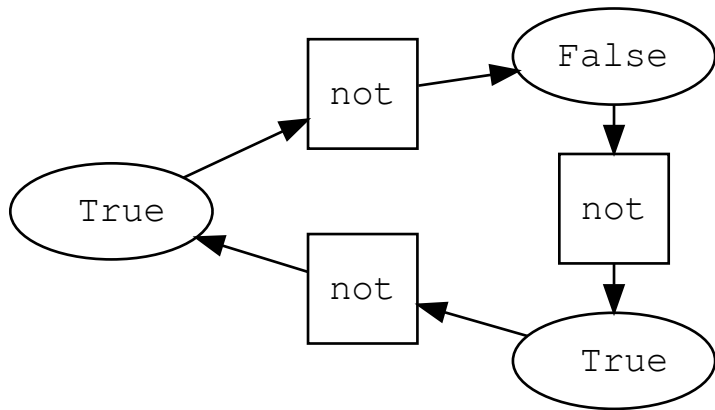


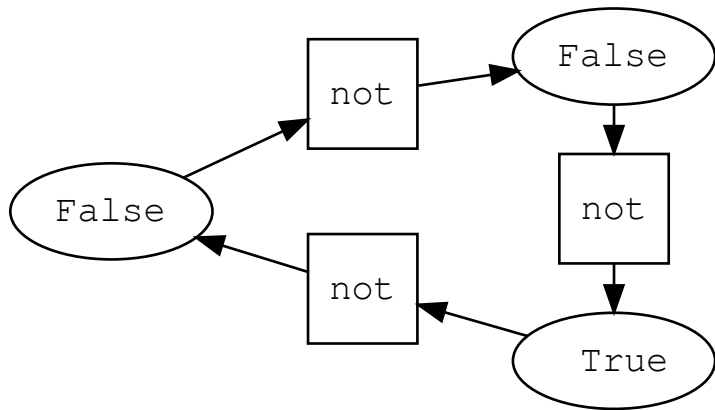


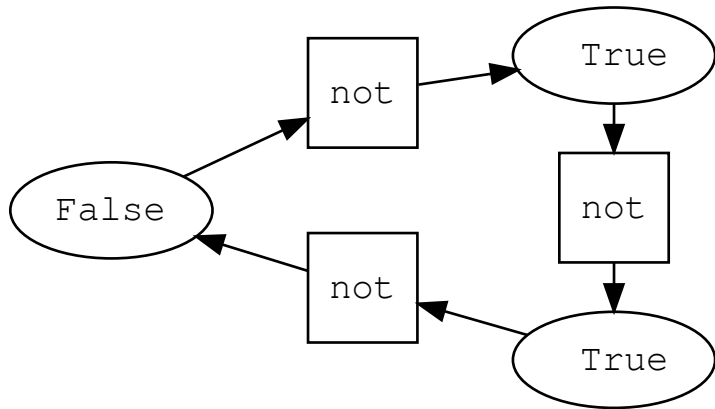


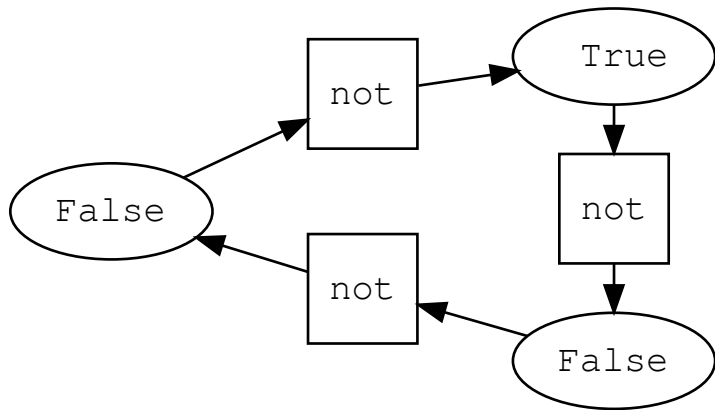


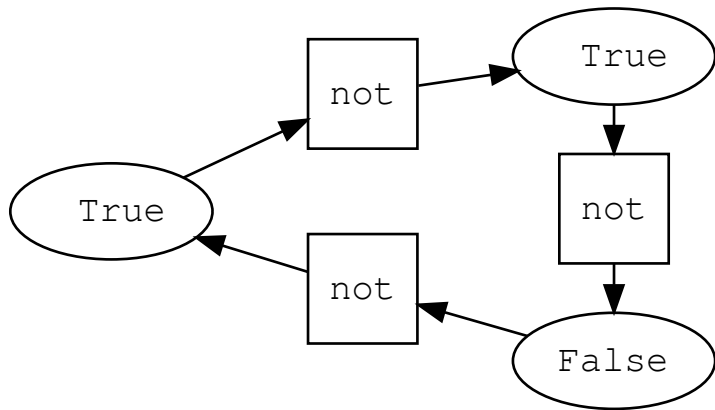


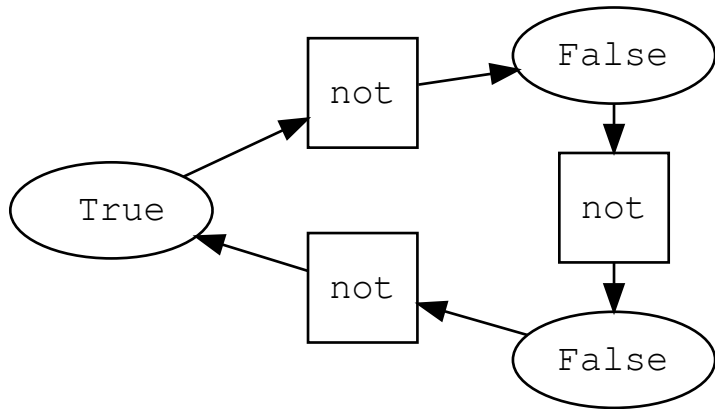












?!

How can we fix this?



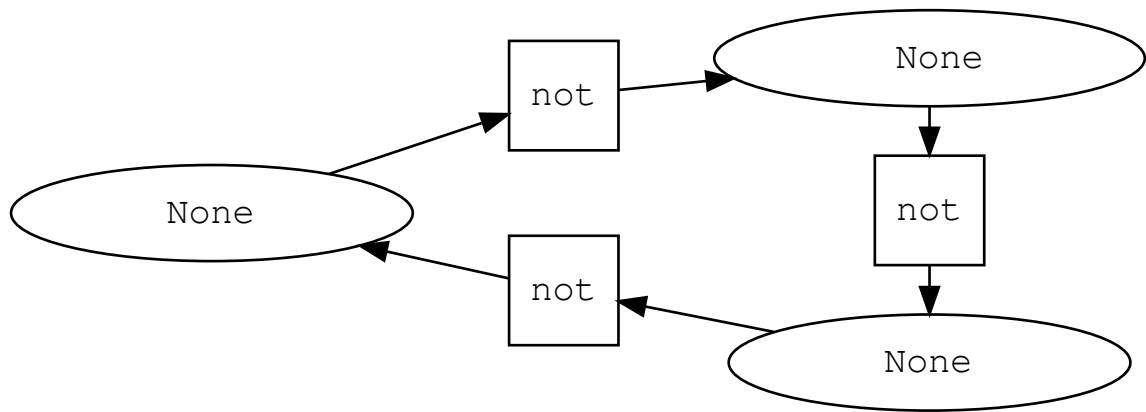
```
data WriteOnce a
  = None
  | Written a
  | TooMany
```

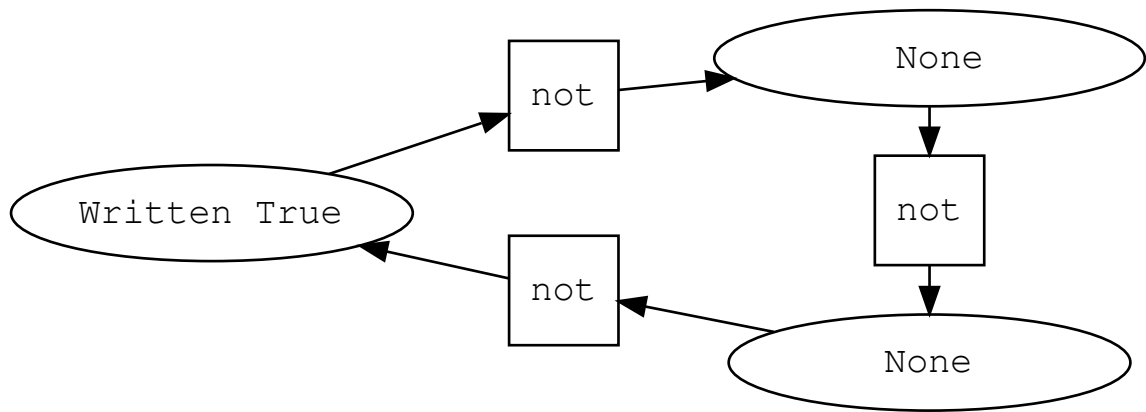
```
data WriteOnce a
  = None
  | Written a
  | TooMany
```

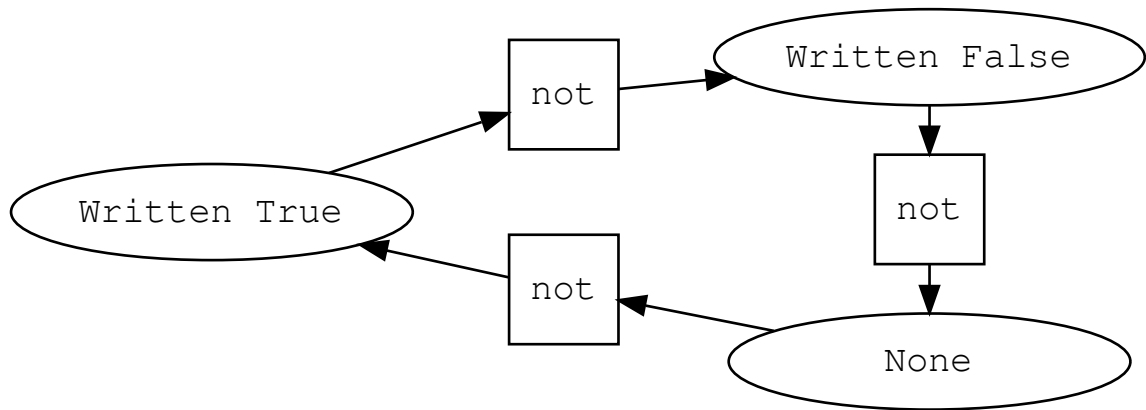
```
tryWrite :: a -> WriteOnce a -> WriteOnce a
tryWrite a w = case w of
  None      -> Written a
  Written b -> TooMany
  TooMany   -> TooMany
```

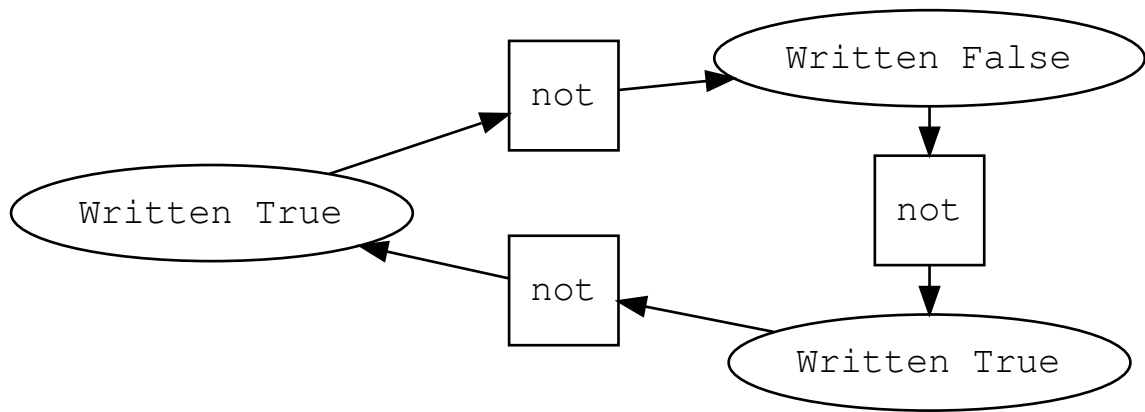
```
data WriteOnce a
  = None
  | Written a
  | TooMany
```

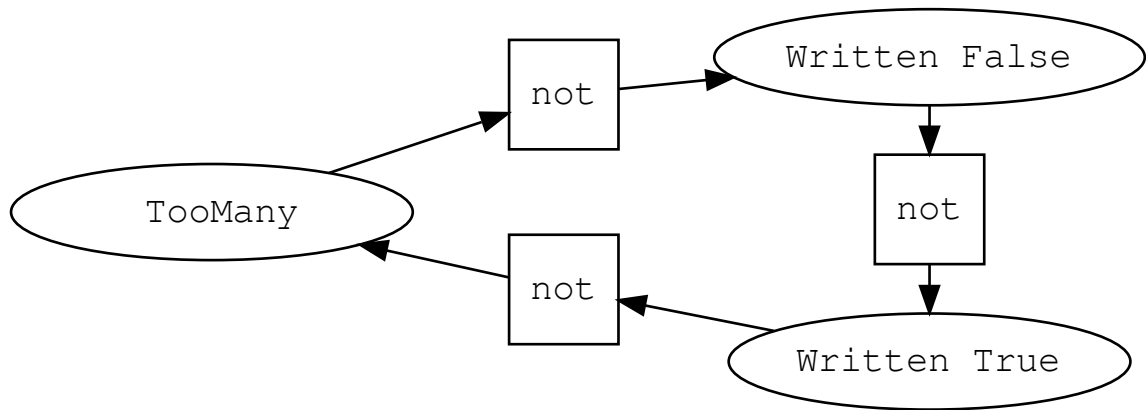
```
tryWrite :: (Eq a) => a -> WriteOnce a -> WriteOnce a
tryWrite a w = case w of
  None      -> Written a
  Written b -> if a == b then Written b else TooMany
  TooMany   -> TooMany
```



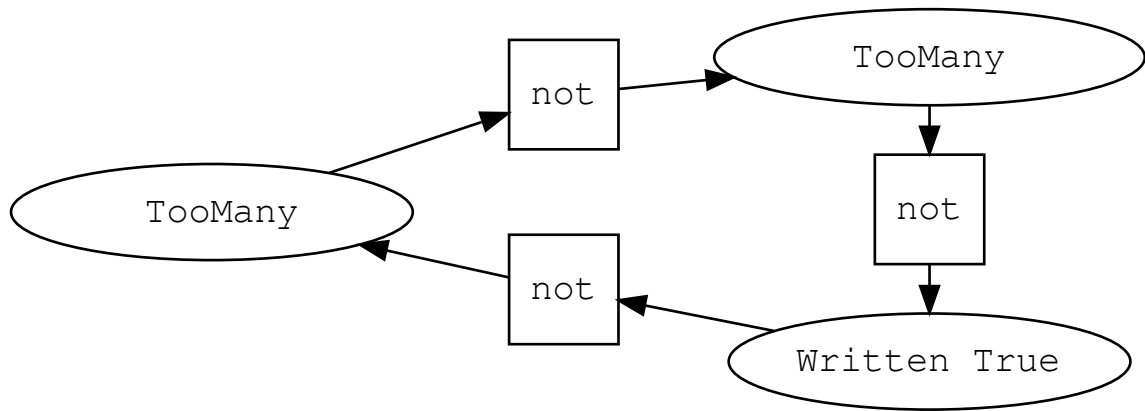


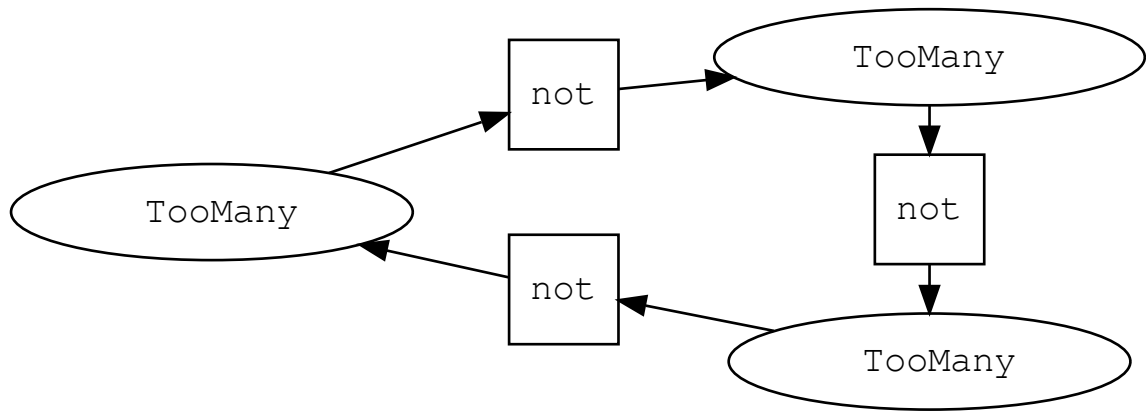










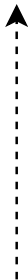


Mutability is **chaos**

WriteOnce is **rigid**

Accumulate information about a value

More information



Less information

-- *I have heard contradictory answers!*

**TooMany**

-- *I know the answer exactly*

**Written** a

-- *I don't know anything*

**None**

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

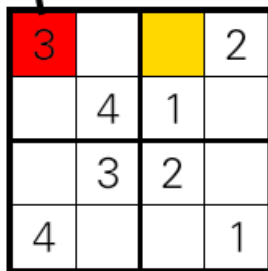
$\{1,3,4\}$

3			2
	4	1	
	3	2	
4			1

3			2
	4	1	
	3	2	
4			1

$\{2,3,4\}$

$\{1,2,4\}$



3		2	2
	4	1	
	3	2	
4			1

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

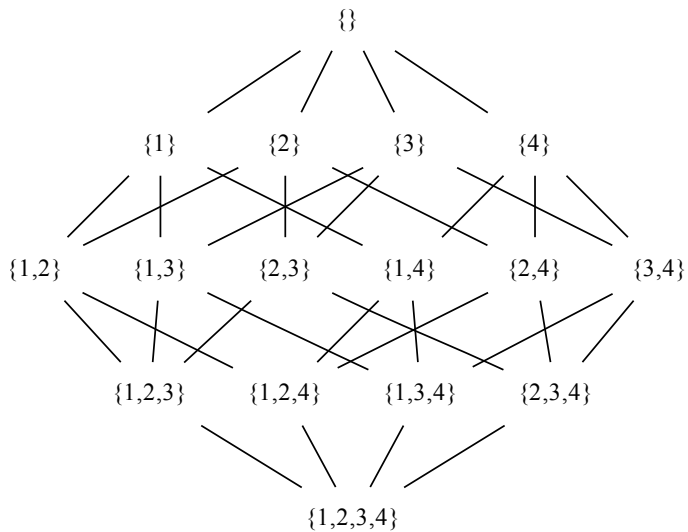
3			2
	4	1	
	3	2	
4			1

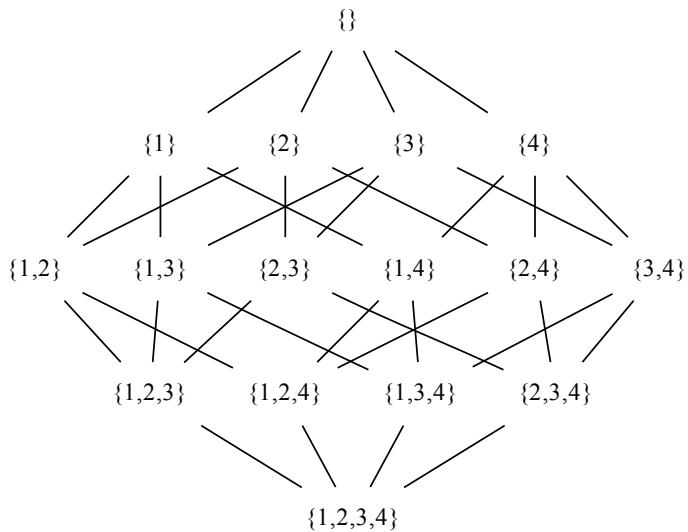


{4}

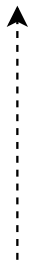
3			2
	4	1	
	3	2	
4			1

3		4	2
	4	1	
	3	2	
4			1

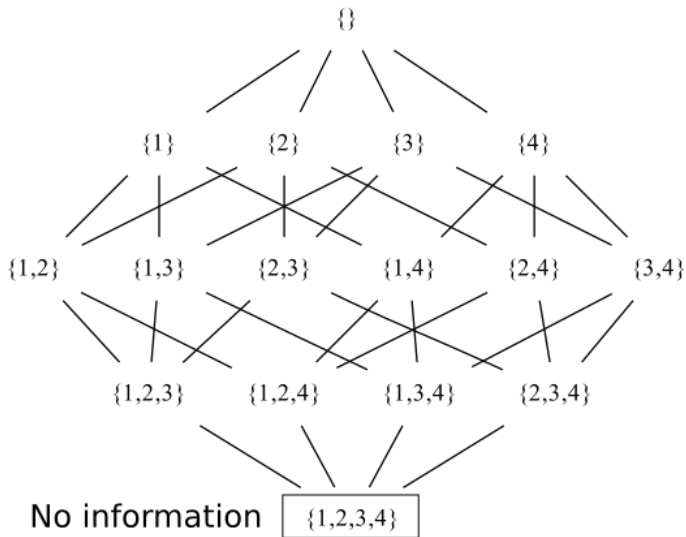




More information



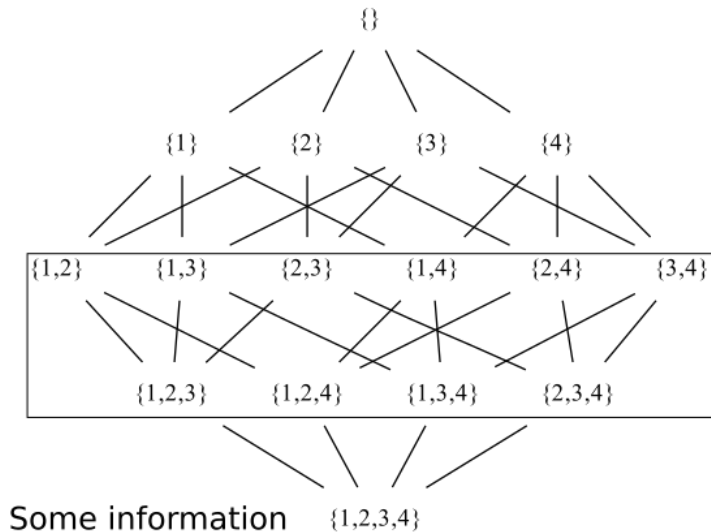
Less information

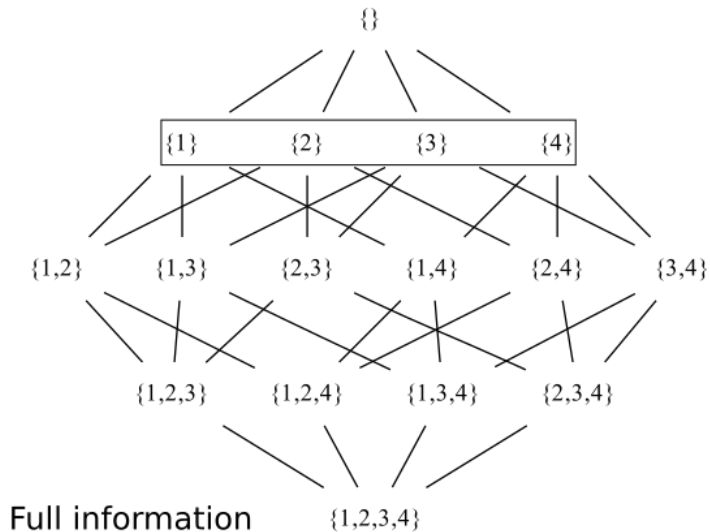


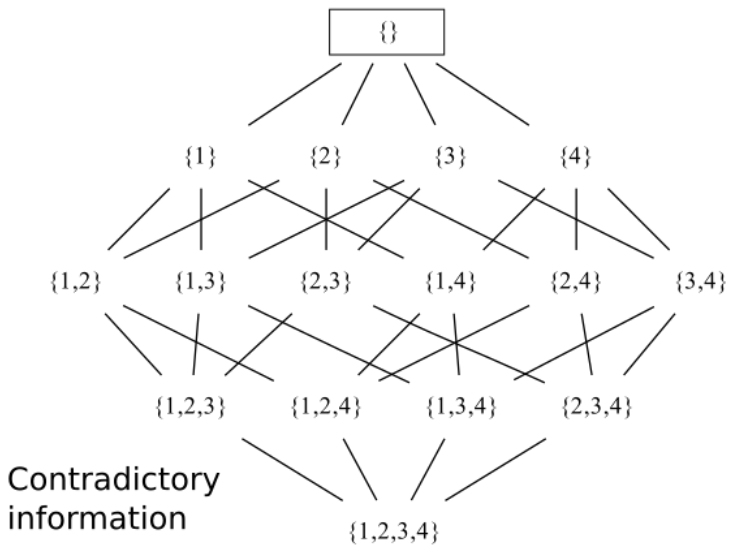
More information



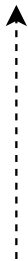
Less information





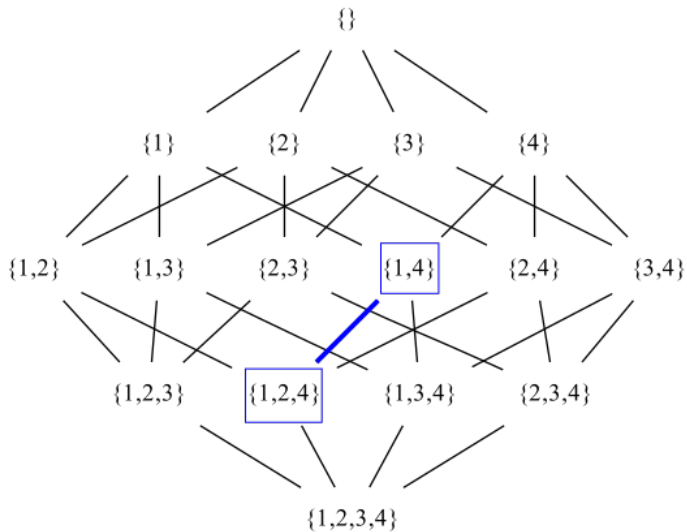


More information

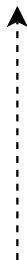


Less information



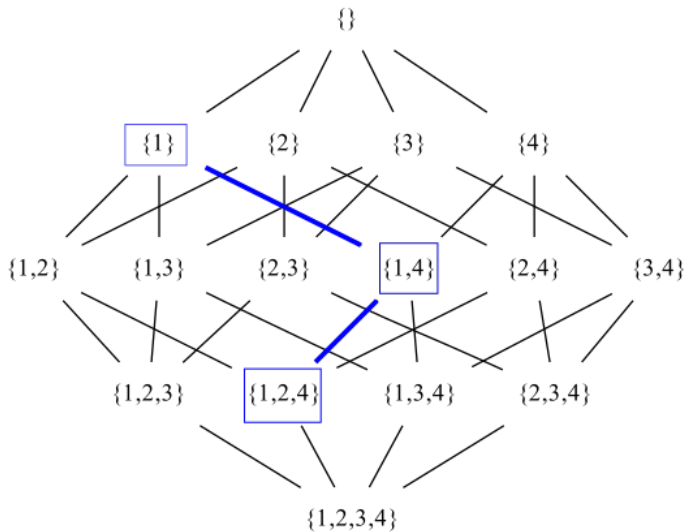


More information

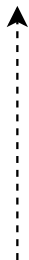


Less information

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

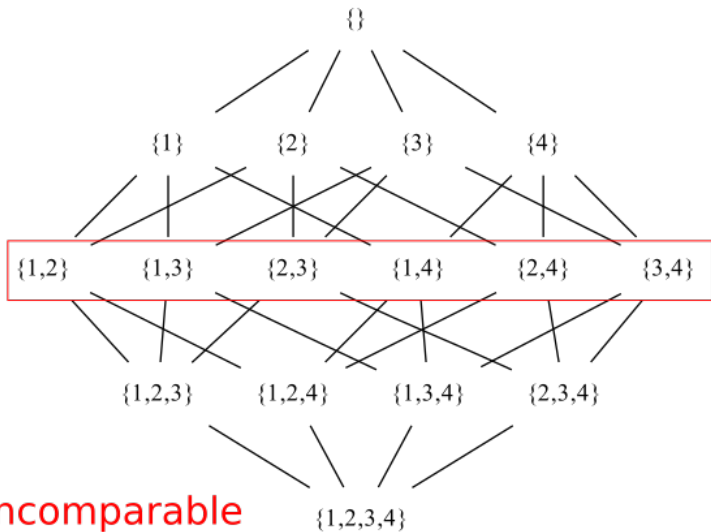


More information

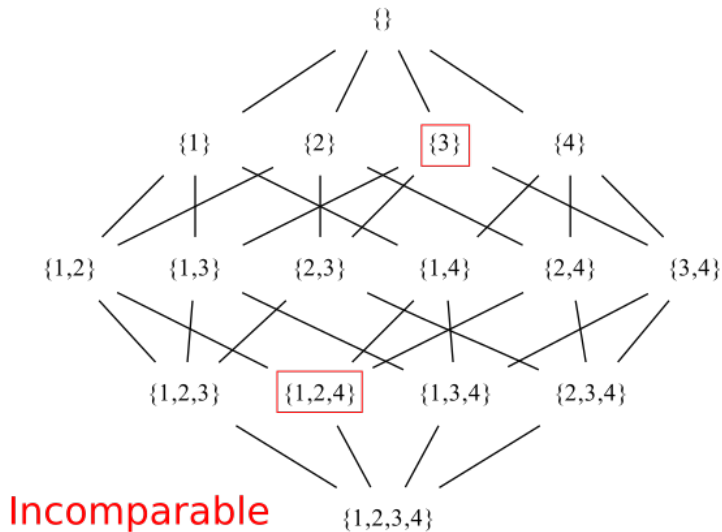


Less information

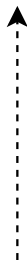
$$\{1,2,4\} < \{1,4\} < \{1\}$$



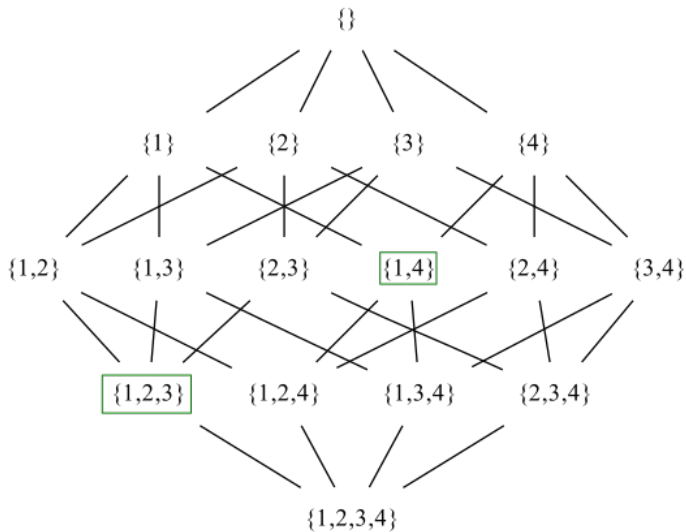
Incomparable



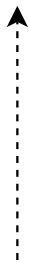
More information



Less information

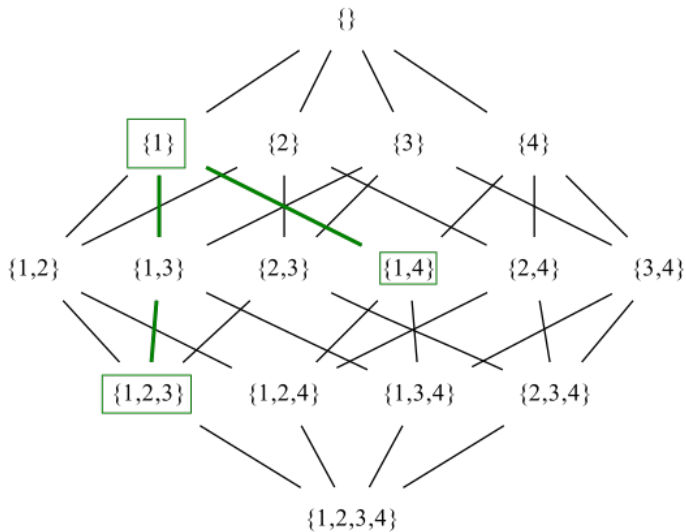


More information

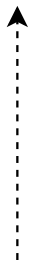


Less information

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



More information



Less information

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

# Bounded join semilattice

Identity:

$$x \vee \textit{bottom} = \textit{bottom} = \textit{bottom} \vee x$$

Associative:

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

Commutative:

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

Idempotent:

$$x \vee x = x$$

```
class SemiLattice a where
  (\\)    :: a -> a -> a
  bottom :: a
```



```
class SemiLattice a where
```

```
  (\\)      :: a -> a -> a
```

```
  bottom :: a
```

```
data SudokuVal = One | Two | Three | Four
```

```
  deriving (Eq, Ord)
```

```
data Possibilities = P (Set SudokuVal)
```

```
class SemiLattice a where
```

```
  (\\)    :: a -> a -> a
```

```
  bottom :: a
```

```
data SudokuVal = One | Two | Three | Four
```

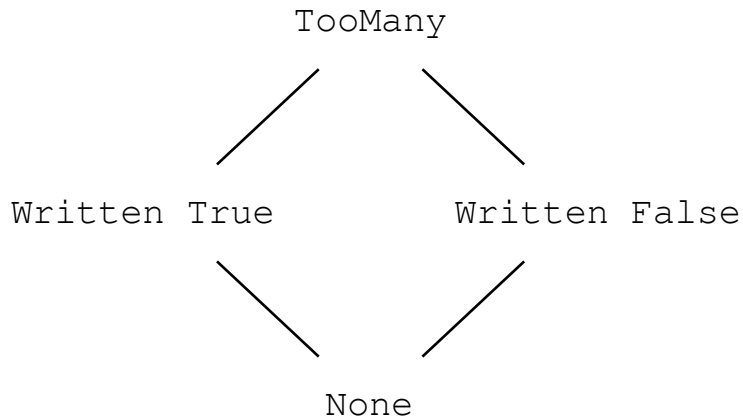
```
  deriving (Eq, Ord)
```

```
data Possibilities = P (Set SudokuVal)
```

```
instance Semilattice Possibilities where
```

```
  P p \\ P q = P (Set.intersection p q)
```

```
  bottom = P (Set.fromList [One, Two, Three, Four])
```



More information



Less information

Cells hold semilattices  
Propagators join information in

WriteOnce

Sets (intersection or union)

Intervals

Search

Unification

many more

# Thanks for listening!

Working code for all these examples and more:

<https://github.com/qfpl/propagator-examples>

## References

Art of the propagator:

<https://dspace.mit.edu/handle/1721.1/44215>

Alexey Radul's PhD Thesis:

<https://dspace.mit.edu/handle/1721.1/54635>

Edward Kmett at Boston Haskell:

<https://www.youtube.com/watch?v=DyPzPeOPgUE>

George Wilson on semi-lattices:

<https://www.youtube.com/watch?v=VXl0EEed8IcU>

## Implementations

Fancy experimental implementation:

<https://github.com/ekmett/guanxi>

Propagators in Haskell

<https://github.com/ekmett/propagators>

Propagators in Clojure:

<https://github.com/tgk/propaganda>