

An Intuition for Propagators

George Wilson

CSIRO's Data61

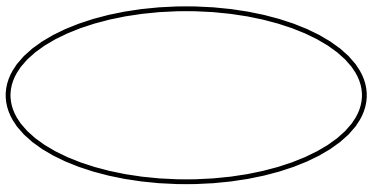
george.wilson@data61.csiro.au

2nd September 2019



1970s, MIT

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

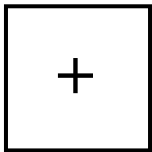


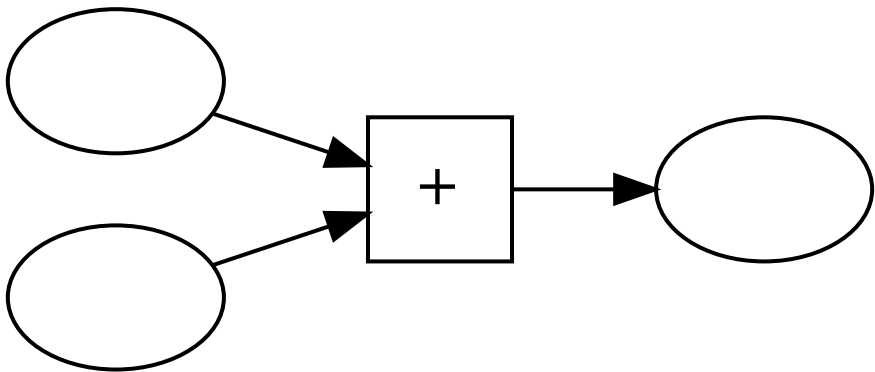


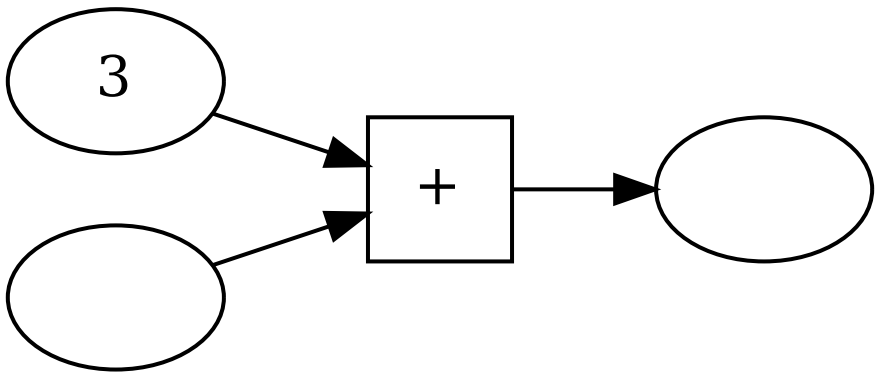
"Hello"

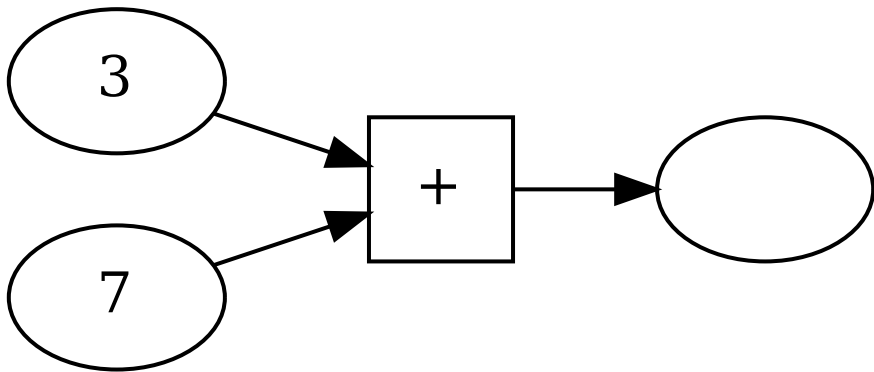


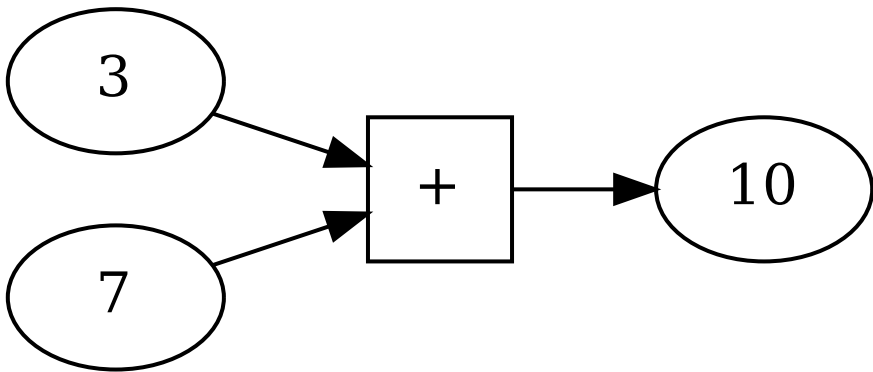
"Compose"

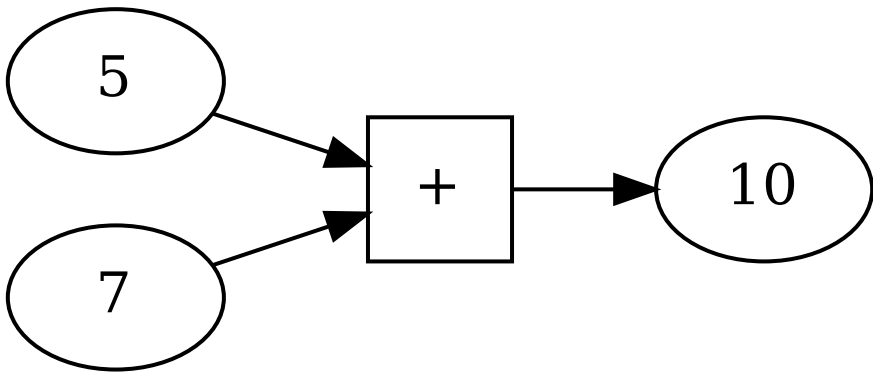


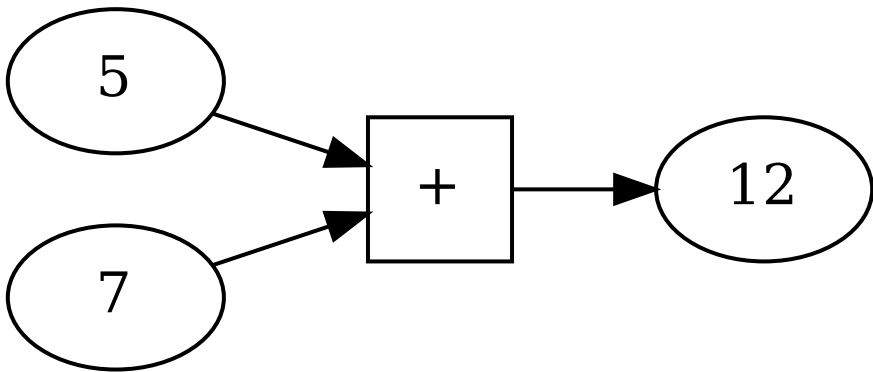












```
-- types
```

```
data Cell a
```

```
data Par a
```

```
instance Monad Par
```

```
-- types
```

```
data Cell a
```

```
data Par a
```

```
instance Monad Par
```

```
-- Creating a cell
```

```
cell      :: Par (Cell a)
```

```
-- types
```

```
data Cell a
```

```
data Par a
```

```
instance Monad Par
```

```
-- Creating a cell
```

```
cell      :: Par (Cell a)
```

```
-- Working with Cells
```

```
content  :: Cell a -> Par (Maybe a)
```

```
write    :: Cell a -> a -> Par ()
```


-- types

data Cell a

data Par a

instance Monad Par

-- Creating a cell

cell :: **Par** (**Cell** a)

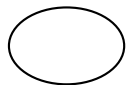
-- Working with Cells

content :: **Cell** a -> **Par** (**Maybe** a)

write :: **Cell** a -> a -> **Par** ()

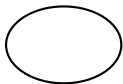
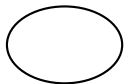
-- Creating a propagator

watch :: **Cell** a -> (a -> **Par** ()) -> **Par** ()



do

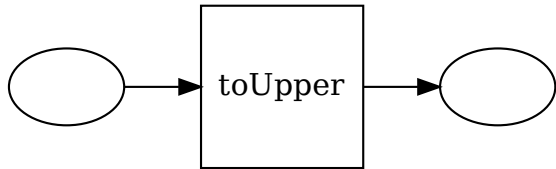
input <- cell



do

input <- cell

output <- cell

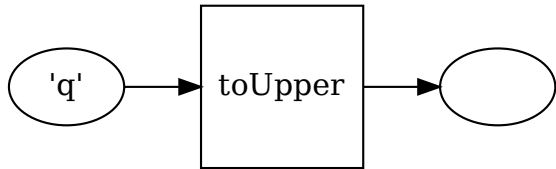


do

```
input  <- cell
```

```
output <- cell
```

```
watch input (\c ->  
  write output (toUpper c))
```



do

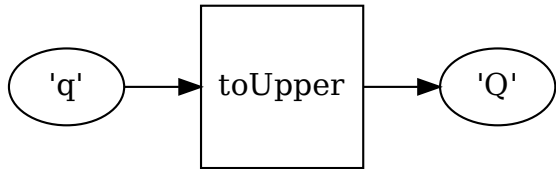
```
input  <- cell
```

```
output <- cell
```

```
watch input (\c ->  
  write output (toUpper c))
```

```
write input 'q'
```

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



do

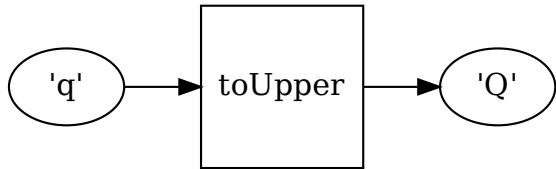
```
input  <- cell
```

```
output <- cell
```

```
watch input (\c ->  
  write output (toUpper c))
```

```
write input 'q'
```

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



do

```
input  <- cell
```

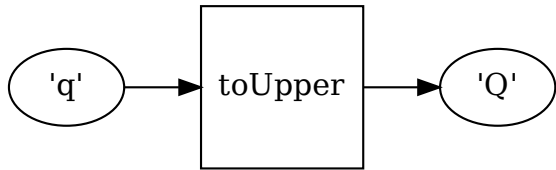
```
output <- cell
```

```
watch input (\c ->  
  write output (toUpper c))
```

```
write input 'q'
```

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

```
lift :: (a -> b) -> Cell a -> Cell b -> Par ()  
lift f input output =  
  watch input (\a ->  
    write output (f a))
```

do

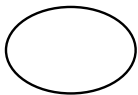
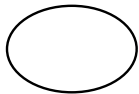
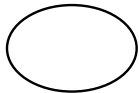
input <- cell

output <- cell

lift toUpper input output

write input 'q'

content output -- Just 'Q'

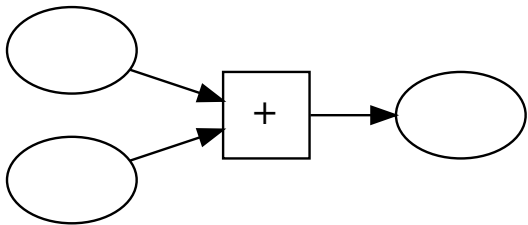


do

inL <- cell

inR <- cell

out <- cell



do

inL <- cell

inR <- cell

out <- cell

watch inL (\x -> do

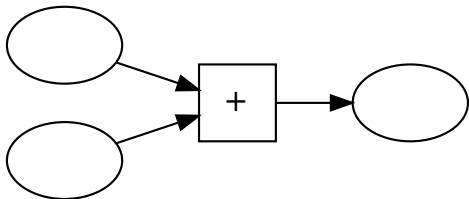
maybeY <- content inR

case maybeY of

Nothing -> pure ()

Just y -> write out (x+y)

```
with :: Cell a -> (a -> Par ()) -> Par ()  
with theCell callback = do  
  maybeA <- content theCell  
  case maybeA of  
    Nothing -> pure ()  
    Just a   -> callback a
```



do

inL <- cell

inR <- cell

out <- cell

watch inL (\x ->

with inR (\y ->

write out (x+y)

```
lift2 :: (a -> b -> c)
      -> Cell a -> Cell b -> Cell c
      -> Par ()
```

```
lift2 :: (a -> b -> c)
      -> Cell a -> Cell b -> Cell c
      -> Par ()
```

```
lift2 f inL inR out = do
  watch inL (\a ->
    with inR (\b ->
      write out (f a b)))
  watch inR (\b ->
    with inL (\a ->
      write out (f a b)))
```

?!

Thanks for listening!