Circuits with Arrows

Ted Cooper
theod@pdx.edu
CS510 – Winter 2016

We'll need delay to simulate gate delays. Let's redefine stream function arrows and introduce an ArrowCircuit class that supports delay.

```
{-# LANGUAGE Arrows #-}
module Circuits where

import Control.Arrow
import Control.Category(Category, (⋙), (.), id)
import Data.List hiding(or)
import Prelude hiding((.), id, or)

newtype SF a b = SF { runSF :: [a] → [b] }

instance Category SF where
  id            = arr id
  SF f ∘ SF g   = SF (f ∘ g)

instance Arrow SF where
  arr f        = SF (map f)
  first (SF f) = SF (unzip ⋙ first f ⋙ uncurry zip)

class ArrowLoop a ⇒ ArrowCircuit a where
  delay :: b → a b b

instance ArrowLoop SF where
  loop (SF f) = SF $ λas →
      let (bs, cs) = unzip (f (zip as (stream cs))) in bs
    where stream ~(x:xs) = x : stream xs

instance ArrowCircuit SF where
  delay x = SF (init ∘ (x :))
```

Now let's build some logic gates.

```
or :: Arrow a ⇒ a (Bool,Bool) Bool
or = arr $ uncurry (||)

nor :: Arrow a ⇒ a (Bool,Bool) Bool
nor = or ⋙ arr not
```

```
flipflop :: ArrowCircuit a ⇒ a (Bool,Bool) (Bool,Bool)
flipflop = loop (arr (λ((a,b),˜(c,d)) → ((a,d),(b,c)))
       ⋙ nor ∗∗∗ nor        -- flip the flop
       ⋙ delay (False,True) -- initialize c low, d high
       ⋙ arr id &&& arr id) -- duplicate output for feedback

-- detect rising edges
edge :: SF Bool Bool
edge = arr id &&& delay False
   ⋙ arr (λ(a,b) → a && not b)

class Signal a where
  showSignal :: [a] → String

instance Signal Bool where
  showSignal bs = concat top++"λn"++concat bot++"λn"
    where (top,bot) = unzip (zipWith sh (False:bs) bs)
          sh True True = ("__","  ")
          sh True False = ("  ","|_")
          sh False True = (" _","| ")
          sh False False = ("  ","__")

instance (Signal a,Signal b) ⇒ Signal (a,b) where
  showSignal xys = showSignal (map fst xys)
                ++ showSignal (map snd xys)

instance Signal a ⇒ Signal [a] where
  showSignal = concat ∘ map showSignal ∘ transpose

sig = concat ∘ map (uncurry replicate)

flipflopInput = sig
  [(5,(False,False)),(2,(False,True)),(5,(False,False)),
  (2,(True,False)),(5,(False,False)),(2,(True,True)),
  (6,(False,False))]

-- to test: putStrLn $ "input:λn" ++ showSignal flipflopInput ++ "output:λn" ++ (
    showSignal $ runSF flipflop flipflopInput)
```