```haskell
-- * Types

-- | Time is rational
type Time = Rational

-- | A time arc (start and end)
type Arc = (Time, Time)

-- | The second arc (the part) should be equal to or fit inside the
-- first one (the whole that it's a part of).
type Part = (Arc, Arc)

-- | An event is a value that's active during a timespan
type Event a = (Part, a)

data State = State {arc :: Arc,
                    controls :: ControlMap
                   }

-- | A function that represents events taking place over time
type Query a = (State -> [Event a])

-- | Also known as Continuous vs Discrete/Amorphous vs Pulsating etc.
data Nature = Analog | Digital
            deriving Eq

-- | A datatype that's basically a query, plus a hint about whether
its events
-- are Analogue or Digital by nature
data Pattern a = Pattern {nature :: Nature, query :: Query a}

data Value = VS { svalue :: String }
           | VF { fvalue :: Double }
           | VI { ivalue :: Int }
           deriving (Eq,Ord,Typeable,Data)

type ControlMap = Map.Map String Value
type ControlPattern = Pattern ControlMap
```