```
from collections import namedtuple
import numpy as np
import tensorflow_probability.substrates.numpy as tfp


tfd = tfp.distributions

X = tfd.Sample(
    tfd.Normal(loc=0., scale=1.),
    sample_shape=(2,),
)
x = X.sample()
```

```
def _to_list(x):
    if isinstance(x, list):
        return x
    else: return [x]

class Monad:
    """Turn a function into a Monad"""

    def __init__(self, fn):
        """The monad 'unit' function"""
        self.__fn = fn  # Make private

    def __call__(self, current_state):
        return self.__fn(current_state)

    def __rshift__(self, rhs_fn):
        """The monad 'bind' operator"""
        def fn(current_state):
            state1, info1 = self(current_state)
            state2, info2 = rhs_fn(state1)
            return state2, _to_list(info1) + _to_list(info2)
        return Monad(fn)

```

```
# Create a MWG sampler of two RWMH
```

```

RwmhInfo = namedtuple("RwmhTuple", ["is_accepted"])

def rwmh(proposal_fn, target_log_density):
    """This is a basic Metropolis Hastings framework that wil
       with any proposal of signature

       proposal_fn :: current_state -> current_state

       where

       target_log_density :: current_state -> floatX
    """

    @Monad
    def kernel(current_state):
        new_state = proposal_fn(current_state)

        log_accept = target_log_density(new_state) - target_l

        if log_accept < np.log(tfd.Uniform(0, 1).sample()):
            return new_state, RwmhInfo(True)

        return current_state, RwmhInfo(False)

    return kernel


def partial_proposal(varnum, proposal):

    def fn(current_state):
        state = current_state.copy()
        partial_state = state[varnum]

        new_partial_state = proposal(partial_state)

        state[varnum] = new_partial_state
        print(f"propose {varnum}")
        return state

```

```
    return fn


def random_walk(current_state):
    return tfd.Normal(loc=current_state, scale=1.0).sample()


kernel0 = rwmh(partial_proposal(0, random_walk), X.log_prob)
kernel1 = rwmh(partial_proposal(1, random_walk), X.log_prob)
kernel2 = rwmh(partial_proposal([0,1], random_walk), X.log_pr
```

```
kernel = kernel0 >> kernel1 >> kernel2
```

```
kernel(x)
```

```
propose 0
propose 1
propose [0, 1]
(array([-0.79854363, -0.84340394], dtype=float32),
 [RwmhTuple(is_accepted=False),
  RwmhTuple(is_accepted=False),
  RwmhTuple(is_accepted=False)])
```