Erratum for Replicate, Reuse, Repeat: Capturing Non-Linear Communication via Session Types and Graded Modal Types

Daniel Marshall

In Section 5.1 of the original paper, the type signature for `forkNonLinear` is given as:

```
forkNonLinear : ∀ {p : Protocol, s : Semiring, r : s} . {SingleAction p}
           ⇒ ((LChan p) [r] → ()) → (LChan (Dual p)) [r]
```

The grading in this type signature is slightly too general, and this allows us to write programs which result in deadlocks. To illustrate this, consider the following example:

```
noSend : ∀ {n : Nat, a : Type}
        . (LChan (Send a End)) [0..1] → ()
noSend [c] = ()

recvOne : ∀ {n : Nat, a : Type}
        . (LChan (Recv a End)) [0..1] → a
recvOne [c] = let (x, c') = recv c; () = close c' in x

main : Int
main = recvOne (forkNonLinear (λc → noSend c))
```

Here, as in the `fromMaybe` example from the paper, the grading `[0..1]` allows us to use both the sender and receiver channels either zero times or one time. We discard the sender channel, using it zero times, but attempt to use the receiver channel a single time in order to receive a value of type `a`. This value never arrives, leading to a deadlock.

To avoid this situation, we introduce an additional `ExactSemiring` constraint into the type signature of `forkNonLinear`, as follows:

```
forkNonLinear : ∀ {p : Protocol, s : Semiring, r : s} . {SingleAction p, ExactSemiring s}
           ⇒ ((LChan p) [r] → ()) → (LChan (Dual p)) [r]
```

This constraint only allows us to use semirings with no *approximation*, which in practice means that both ends of the channel have to be used with truly identical grades. As far as semirings mentioned in the paper are concerned, this means that the grade `r` here is allowed to be a natural number, representing exact usage, but importantly is *not* allowed to be an interval, representing a range of possible usages.