Problems with The Lazy Queue

The lazy queue operations are sometimes expensive because rotation is done monolithically:

```
... fs ++ reverse bs ...
```

To achieve $O(1)$ worst-case time bound, we need to:

- do rotation incrementally
- schedule to evaluate it often

This spreads out the cost of the expensive operation.

Incremental Rotation

Rather than relying on `reverse` to rotate, we write our own rotation:

```
rotate [] (y:[]) a     = y:a
rotate (x:xs) (y:ys) a = x : rotate xs ys (y:a)
```

(We ignore `rotate [] y:ys a` with `ys /= []`. We will make sure it never happens.)

So `rotate xs ys []` has the same value as `xs ++ reverse ys`, except that it is incrementally lazy. The parameter a acts as a kind of accumulator.

More generally, `rotate xs ys a == xs ++ reverse ys ++ a`.

$O(1)$ Lazy Queue

The queue has a front list, a back list, and a schedule:

```
data SQueue a = SQ [a] [a] [a]
```

The schedule holds an unevaluated `rotate` expression. It is usually a suffix of the front list. By evaluating it once in a while, we discharge the rotation incrementally.

```
snoc (SQ f b s) x   = exec f (x:b) s
head (SQ (h:f) b s) = h
tail (SQ (h:f) b s) = exec f b s
```

Here `exec` evaluates the schedule once and returns the new queue.

Scheduling

`exec` evaluates the schedule by matching it against patterns. It also forgets the head so that the rest gets evaluated next time.

If the schedule becomes empty, we simply create a new rotation as the new schedule (and as the new front list). It is this time when the back list is emptied.

```
exec f b (x:s) = SQ f b s
exec f b []    = SQ f' [] f' where f' = rotate f b []
```

Now each operation takes $O(1)$ time.