Now each operation takes O(1) time.

execute f p [] where f = rotate f p
execute f p (x:p) = f (cons x (f p))

empyed.

Schedule (and so the new front list). It is this time when the back list is
If the schedule becomes empty, we simply create a new rotation as the new
The head so that the rest gets evaluated next time.
execute evaluates the schedule by matching all adjacent patterns. It also forgets
Schedule

More generally, rotate xs y = y ++ rotate xs y or a
That is, incrementally lazy. The parameter acts as a kind of accumulation:
 So rotate xs y is the same value as xs ++ rotate xs y, except
 happens.)
We ignore rotate [] y s with y /= s. We will make sure it never
rotate (x:xs) (y:ys) = x : rotate xs (y:ys)
rotate [] y s = y : s

Either then relying on reverse to rotate. we write our own rotation:

Incremental Rotation

Here execute evaluates the schedule once and returns the new queue.

execute f q s = execute f q q
head (sg q) p = execute f q (x:p)
since (sg f q) p q = execute f q (x:p)

incrementally
of the form list by evaluating it once in a while, we discharge the rotation
The schedule holds an unvalued rotated expression. It is usually a suffix

data Queue a = SQ [a] [a]

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

Lazy Queue O(1)

This speeds up the cost of the expensive operation:

- Schedule to evaluate is often
- do rotation incrementally

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

- is ++ reverse ps ...
- monolithic

The lazy queue operations are sometimes expensive because rotation is done

Problems with The Lazy Queue
