A queue, according to our axioms, has an unlimited capacity to have items joined onto it. A limited-queue is a similar data structure but with a limited capacity to have items joined onto it.

(a) Design axioms for a limited-data-queue.
§ I'm introducing new name `full`, which tells whether a queue is full (of course). This allows an implementation in which `full` might say `⊤` for a queue with 3 long items, and `⊥` for another queue with 3 short items in it. It also allows an implementation to allocate more space at any time, and deallocate unused space at any time. I also think it's the easiest solution.

```plaintext
emptyq: queue
full: queue→bin
¬ full q ⇒ join q x: queue
¬ full q ⇒ join q x ≠ emptyq
¬ full q ∧ ¬ full r ⇒ (join q x = join r y = q=r ∧ x=y)
q+emptyq ⇒ leave q: queue
q+emptyq ⇒ front q: X
¬ full emptyq ⇒ leave (join emptyq x) = emptyq
q+emptyq ∧ ¬ full q ⇒ leave (join q x) = join (leave q) x
¬ full emptyq ⇒ front (join emptyq x) = x
q+emptyq ∧ ¬ full q ⇒ front (join q x) = front q
```

(b) Design axioms for a limited-program-queue.
§
```
mkemptyq ⇒ isemptyq'
isemptyq ∧ ¬ isfullq ∧ join x ⇒ front' = x ∧ ¬ isemptyq'
¬ isemptyq ∧ leave ⇒ ¬ isfullq'
¬ isemptyq ∧ ¬ isfullq ∧ join x ⇒ front' = front ∧ ¬ isemptyq'
isemptyq ∧ ¬ isfullq ⇒ (join x. leave = mkemptyq)
¬ isemptyq ∧ ¬ isfullq ⇒ (join x. leave = leave. join x)
```

(c) Can the limit be 0?
§ The limit can be 0. That happens in (a) when `full` is the constant `⊤` function; even `full emptyq` is `⊤`. In (b) it happens when `isfullq` is identically `⊤`. 