Quickcheck: recap

-
  ```
  let rec rev = function
  | [] -> []
  | x :: xs -> append (rev xs, [x])
  ```

-
  ```
  let prop_revIsOrig (xs:int list) =
     rev xs = xs
  ```

-
  ```
  do Check.Quick prop_revIsOrig
     > Falsifiable, after 3 tests (5 shrinks)
     [1; 0]
  ```

Quickcheck: how

- Checking $\forall \mathbf{x} : \boldsymbol{\tau}.\ C(\mathbf{x})$ means trying to see if there is an assignment $\mathbf{x} \to \mathbf{a}$ at type $\boldsymbol{\tau}$ such that $\neg C(\mathbf{a})$ holds
  - e.g. checking ∀xs : `int list. rev xs = xs` means finding xs → [1;0], for which rev xs ≠ xs
- Quickcheck generates *pseudo-random* values up to size *k (EndSize)* and stops when
  - a counterexample is found, or
  - the maximum size of test values has been reached (*MaxTest*), or
  - a default timeout expires (*MaxFail*)

Conditional laws

- More interesting are *conditional laws*:
  - `ordered xs ⟹ ordered (insert x xs)`
- Here we generate random lists that may or may not be sorted and then check if insertion preserves ordered-ness
- If a candidate list does not satisfies the condition it is discarded
  - *Coverage is an issue:* what's the likelihood of randomly generating lists (of length > 1) that are *sorted*?
- Quickcheck gives combinator to *monitor* test data distribution – but in the end one has to write an ad-hoc generator, here yielding only ordered lists