let reverse \( \ell = \)

| \([\text{I}] \rightarrow [\text{I}]\) |
| \(x::xs \rightarrow \text{append}(\text{reverse } xs, [x])\)

let append \((\ell_1, \ell_2) = \)

| \([\text{I}] \rightarrow \ell_2\) |
| \(x::xs \rightarrow x::(\text{append}(xs, \ell_2))\)

proof for append:

want: \(\text{append}(\ell_1, \ell_2) = [x_1, \ldots, x_n, y_1, \ldots, y_m]\) when \(\ell_1 = [x_1, \ldots, x_n]\) and \(\ell_2 = [y_1, \ldots, y_m]\)

base case: \(n = 0\) \(\rightarrow\) \(\text{append}(\ell_1, \ell_2) = \ell_2 = [y_1, \ldots, y_m]\)

inductive step: assume append works for \(|\ell_1| \leq n\), prove for \(|\ell_1| = n + 1\)

\(\ell_1 = [x_0, \ldots, x_n] \rightarrow\) so we match to case 2:

\(\ell_2 = [y_1, \ldots, y_m]\)

\(x = x_0\) \(xs = [x_1, \ldots, x_n]\) \n
and \(\text{append}(\ell_1, \ell_2) = x_0::(\text{append}(xs, \ell_2))\)

by 1.H. \(\text{append}(xs, \ell_2) = [x_1, \ldots, x_n, y_1, \ldots, y_m]\)

so \(\text{append}(\ell_1, \ell_2) = x_0::[x_1, \ldots, x_n, y_1, \ldots, y_m]\)

\(= [x_0, x_1, \ldots, x_n, y_1, \ldots, y_m]\) \(\Box\)

proof for reverse:

want: \(\text{reverse } \ell = [x_n, \ldots, x_1]\) if \(\ell = [x_1, \ldots, x_n]\)

base case: \(n = 0\)

\(\ell = [\text{I}] \rightarrow \text{reverse } \ell = [\text{I}]\)

induction step: assume reverse works for \(|\ell| \leq n\), prove for \(|\ell| = n + 1\)

\(\ell = [x_0, x_1, \ldots, x_n] \rightarrow\) \(\text{reverse } \ell = \text{append}(\text{reverse } [x_1, \ldots, x_n], [x_0])\)

by induction hypothesis (1.H.) \(\text{reverse } [x_1, \ldots, x_n] = [x_n, \ldots, x_1]\)

by proof of append, \(\text{append}(\ell, \ell_2) = [x_0, x_1, \ldots, x_n, x_0]\) \(\Box\)
let rev l = helper (l, [])

let helper (l, acc) =
    match l with
    | [] -> acc
    | x::xs -> helper (xs, x::acc)

proof for rev l:

want: rev l = [x_n, ..., x_1] when [x_1, ..., x_n] = l

to show this, we need to show that helper ([x_1, ..., x_n], [y_1, ..., y_m]) = [x_n, ..., x_1, y_1, ..., y_m]

base case: n = 0
helper ([], [y_1, ..., y_m]) -> [y_1, ..., y_m]

inductive step: i.h.: rev k works correctly for |k| < n

helper ([x_0, x_1, ..., x_n], [y_1, ..., y_m]) -> helper ([x_1, ..., x_n], [x_0, y_1, ..., y_m])

by i.h., this = [x_n, ..., x_1, x_0, y_1, ..., y_m] □

Russian peasant exponentiation

recursive method to compute b^e

rpe (b, e) =
    if (e = 0) then 1
    elif (b = 0) then 0  // catches special case
    elif (e is odd) then (b * rpe (b, e-1))
    else  // e is even
        let a = rpe (b, e/2) in a * a

proof for rpe:
want to show rpe (b, e) = b^e for all e ≥ 0

base case: e = 0
rpe (b, 0) = 1

inductive step: Assume rpe (b, e) works correctly for e < n

if e is odd, rpe (b, e) = (b * rpe (b, e-1))
    by i.h. rpe (b, e-1) = b^(e-1)
    then b * b^(e-1) = b^e

if e is even, rpe (b, e) = (rpe (b, e/2) * rpe (b, e/2))
    by i.h. rpe (b, e/2) = b^(e/2)
    then b^(e/2) * b^(e/2) = b^e □
Another way to compute exponents:

\[
\text{fastexp} \left( b, e \right) = \\
\text{let helper} \left( b, e, a \right) = \\
\quad \text{if } \left( e = 0 \right) \text{ then } a \\
\quad \cdot \text{ else helper} \left( b, e - 1, b \times a \right) \\
\quad \cdot \text{ else helper} \left( b, e, b^2, a \right) \\
\quad \text{if } b = 0 \text{ then } 0 \cdot \text{ else helper} \left( b, e, 1 \right)
\]

How can we reason about this program? We are no longer making recursive calls using strictly decreasing arguments, so we can't use the same technique we applied previously.

New idea: use an invariant. This is a quantity that remains constant after each step.

For \text{fastexp}, we will define our invariant to be \( I = b^e \cdot a \)

proof: No matter what branch we take in \text{helper}, \( I \) remains constant.

\( e \) is odd:

\[
\text{helper} \left( b_0, e_0, a_0 \right) \rightarrow \text{helper} \left( b_0, e_0 - 1, b_0 \times a \right) \\
\quad b_n \quad e_n \quad a_n
\]

w.t.s. \( b_0 \cdot b_0 \cdot a_0 = b_0^{e_0} \cdot a_0 \\
\quad = b_0^{e_0 - 1} \cdot (b_0 \cdot a_0) \\
\quad = b_0^{e_0} \cdot a_0 \quad \checkmark
\]

\( e \) is even:

\[
\text{helper} \left( b_0, e_0, a_0 \right) \rightarrow \text{helper} \left( b_0, e_0, b_0^{e_0 / 2}, a_0 \right) \\
\quad b_n \quad e_n \quad a_n
\]

w.t.s. \( b_0 \cdot a_0 = b_0^{e_0} \cdot a_0 \\
\quad = (b_0 \cdot b_0)^{e_0 / 2} \cdot a_0 \\
\quad = b_0^{e_0} \cdot a_0 \quad \checkmark \quad \Box \quad I \text{ is an invariant}
\]

proof that \text{fastexp} is correct; w.t.s.: \( \text{fastexp} \left( b, e \right) = b^e \)

\( \text{if } b = 0, \quad \text{fastexp} \left( 0, e \right) = 0 \checkmark \)

\( \text{if } b > 0, \quad \text{fastexp} \left( b, e \right) = \text{helper} \left( b, e, 1 \right) \)

so in the initial call, \( I = b^e \)

by examining the code, we see \text{helper} ends when \( e = 0 \), outputting \( a_f \). Does \( a_f = b^e \) at this point?

We showed \( I \) remains constant, so

\[
b^e = b_f^0 \cdot a_f \quad \text{(where } b_f \text{ denotes } "e \text{ final"}) \\
\quad = 1 \cdot a_f = a_f \quad \checkmark \quad \Box \quad \text{fastexp} \left( b, e \right) = b^e
\]

Is we proved that the value of \( I \) remains constant at each step, so the value of \( I \) at the beginning must equal the value of \( I \) when helper terminates,

initially \( I = b^e \cdot 1 \quad \checkmark \quad \text{at the end } I = b_f^0 \cdot a_f = 1 \cdot a_f \)

\[
b^e = b^e \quad \checkmark
\]

\( a_f \) is our output and \( a_f = b^e \checkmark \)