Homework

• (easy) Finish the stack example by giving an implementation and proof outline for pop.

• (easy) Write out the proof outline for reverse in the functional version.

• (easy to medium) Specify and verify the following program copy takes a pointer to a binary tree, and copies it into a separate heap chunk, returning the pointer to the copy.

\[
\text{copy } (p : \text{ptr}) = \\
\text{if } p = \text{null} \text{ then return null} \\
\text{else } v \leftarrow p.\text{value}; \ tl \leftarrow p.\text{left}; \ tr \leftarrow p.\text{right}; \\
\text{p’} \leftarrow \text{alloc } v \text{ null null; } \\
\text{tl’} \leftarrow \text{copy } tl; \ v’.\text{left} := \text{tl’}; \\
\text{tr’} \leftarrow \text{copy } tr; \ v’.\text{right} := \text{tr’}; \\
\text{return } p’
\]

• (difficult) Specify and verify a union-find data structure. It consists of a number of inverted trees. The nodes in a tree all have a parent field pointing to their parent.

It exports the following methods.

– find(x) returns a root of x, compressing the paths along the way.

\[
\text{find } (x : \text{ptr}) = \\
i \leftarrow ! (x.\text{parent}) ; \\
\text{if } i \neq \text{null} \text{ then } j \leftarrow \text{find } i; \ x.\text{parent} := j; \text{ return } j \\
\text{else return } i
\]

– union(x, y) joins the trees of x and y. In practice, structure keeps tree sizes, to join smaller to larger, but we ignore that here.

\[
\text{union } (x \ y : \text{ptr}) = i \leftarrow \text{find } x; \ j \leftarrow \text{find } y; \text{ if } i \neq j \text{ then } i.\text{parent} := j
\]