CIS 705 — Programming Languages — Spring 2010

Assignment 3

Model Answers

Notation

Given a list *xs*, we abbreviate `List.length` *xs* to $|xs|$. And we abbreviate `List.rev` to `rev`.

Exercise 1

Lemma 1.1

(1) For all xs and ys of type `'a list`*, edits of type* `'a edit list`*, and xs′ of type* `'a annot list`*, if all the elements of edits are deletes,* `doEdits`$(edits, xs) = ys$*, all the elements of xs′ are annotated with* `Original`*, and* `annotListToList` $xs' = xs$*, then* `annotListToList`(`doEditsAnnot`$(edits, xs')) = ys$ *and all the elements of* `doEditsAnnot`$(edits, xs')$ *are annotated with* `Original`*.*

(2) For all xs and ys of type `'a list`*, edits of type* `'a edit list`*, and xs′ of type* `'a annot list`*, if all the elements of edits are transposes,* `doEdits`$(edits, xs) = ys$*, all the elements of xs′ are annotated with* `Original` *or* `Transposed`*, and* `annotListToList` $xs' = xs$*, then* `annotListToList`(`doEditsAnnot`$(edits, xs')) = ys$ *and all the elements of* `doEditsAnnot`$(edits, xs')$ *are annotated with* `Original` *or* `Transposed`*.*

(3) For all xs and ys of type `'a list`*, edits of type* `'a edit list`*, and xs′ of type* `'a annot list`*, if all the elements of edits are inserts/appends,* `doEdits`$(edits, xs) = ys$ *and* `annotListToList` $xs' = xs$*, then* `annotListToList`(`doEditsAnnot`$(edits, xs')) = ys$*.*

Proof. Follows from a comparison of: `deleteAnnot` with `delete`; `transposeAnnot` with `transpose`; `insertAnnot` with `insert`; and `appendAnnot` with `append`. □

A value *edits* of type `'a edit list` is *standard* iff there are $edits_1$, $edits_2$ and $edits_3$ of type `'a edit list` such that $edits = edits_1 \mathbin{@} edits_2 \mathbin{@} edits_3$, $edits_1$ contains only deletes, $edits_2$ contains only transposes, and $edits_3$ contains only inserts/appends.

Lemma 1.2

For all xs and ys of type `'a list` *and edits of type* `'a edit list`*, if edits is standard and* `doEdits`$(edits, xs) = ys$*, then* `annotListToList`(`doEditsAnnot`$(edits,$ `listToAnnotList` $xs)) = ys$*.*

Proof. Follows from Lemma 1.1, since `annotListToList`(`listToAnnotList` $xs) = xs$ and all the elements of `listToAnnotList` *xs* are annotated with `Original`. □

Lemma 1.3

For all xs and ys of type `'a list` *and edits of type* `'a edit list`*, if* $|edits| = 2$*, the first element of edits is not a delete, the second element of edits is a delete and* `doEdits`$(edits, xs) = ys$*, then either:*

(1) there is an edits′ of type `'a edit list` *such that* $|edits'| < 2$ *and* `doEdits`$(edits', xs) = ys$*; or*

(2) there is an edits′ of type `'a edit list` *such that* $|edits'| = 2$*, the first element of edits′ is a delete, the second element of edits′ is not a delete,* `doEdits`$(edits', xs) = ys$*, and, for all xs′ and ys′ of type* `'a annot list`*, if* `annotListToList` $xs' = xs$ *and* `doEditsAnnot`$(edits', xs') = ys'$ *(so that* `annotListToList` $ys' = ys$*), then* `doEditsAnnot`$(edits, xs') = ys'$*.*

Proof. Suppose xs and ys have type `'a list`, $edits$ has type `'a edit list`, $|edits| = 2$, the first element of $edits$ is not a delete, the second element of $edits$ is a delete and `doEdits`$(edits, xs) = ys$. There are 8 cases to consider.

- Suppose $edits = [$`Transpose` $n;$ `Delete` $m]$ and $m < n$. Since `doEdits`$(edits, xs) = ys$, it follows that $0 \leq m < n \leq |xs| - 2$. Let $edits' = [$`Delete` $m;$ `Transpose`$(n - 1)]$. Then $|edits'| = 2$, the first element of $edits'$ is a delete, the second element of $edits'$ is not a delete, `doEdits`$(edits', xs) = ys$, and, for all xs' and ys' of type `'a annot list`, if `annotListToList` $xs' = xs$ and `doEditsAnnot`$(edits', xs') = ys'$, then `doEditsAnnot`$(edits, xs') = ys'$. (To see that the last part holds, suppose xs' and ys' have type `'a annot list`, `annotListToList` $xs' = xs$ and `doEditsAnnot`$(edits', xs') = ys'$. Then position m of xs' must have annotation `Original`, and positions n and $n + 1$ must be annotated with `Original` or `Transposed`, since after the deletion of position m, they will be at positions $n - 1$ and n, and will then be transposed, and annotated with `Transposed`. Thus it is easy to see that `doEditsAnnot`$(edits, xs') = ys'$.
- Suppose $edits =$ [`Transpose` $n;$ `Delete` $n]$. Then $0 \leq n \leq |xs| - 2$. Let $edits' =$ [`Delete`$(n+1)$]. Then $|edits'| < |edits|$ and `doEdits`$(edits', xs) = ys$.
- Suppose $edits =$ [`Transpose` $n;$ `Delete`$(n + 1)$]. Then $edits' =$ [`Delete` n] has the property of (1).
- Suppose $edits =$ [`Transpose` $n;$ `Delete` m] and $m > n+1$. Then $edits =$ [`Delete` $m;$ `Transpose` n] has the property of (2).
- Suppose $edits =$ [`Append` $x;$ `Delete` n]. Because `doEdits`$(edits, xs) = ys$, there are two subcases to consider.
  - Suppose $n < |xs|$. Then $edits' =$ [`Delete` $n;$ `Append` x] has the property of (2)
  - Suppose $n = |xs|$. Then $edits' = [\,]$ has the property of (1).
- Suppose $edits =$ [`Insert`$(n, x);$ `Delete` m] and $m < n$. Then $edits' =$ [`Delete` $m;$ `Insert`$(n - 1, x)$] has the property of (2).
- Suppose $edits =$ [`Insert`$(n, x);$ `Delete` n]. Then $edits' = [\,]$ has the property of (1).
- Suppose $edits =$ [`Insert`$(n, x);$ `Delete` m] and $m > n$. There are two subcases to consider.
  - Suppose $n < |xs| - 1$. Then $edits' =$ [`Delete`$(m - 1);$ `Insert`(n, x)] has the property of (2).
  - Suppose $n = |xs| - 1$. Then $edits' =$ [`Delete` $n;$ `Append` x] has the property of (2).

□

Lemma 1.4
For all xs and ys of type `'a list` *and edits of type* `'a edit list`*, if* $|edits| = 2$*, the first element of edits is an insert/append, the second element of edits is a transpose, and* $\texttt{doEdits}(edits, xs) = ys$*, then either:*

(1) there is an $edits'$ *of type* `'a edit list` *such that* $|edits'| < 2$ *and* $\texttt{doEdits}(edits', xs) = ys$*; or*

(2) there is an $edits'$ *of type* `'a edit list` *such that* $|edits'| = 2$*, the first element of* $edits'$ *is a transpose, the second element of* $edits'$ *is an insert/append,* $\texttt{doEdits}(edits', xs) = ys$*, and, for all* xs' *and* ys' *of type* `'a annot list`*, if* $\texttt{annotListToList}\, xs' = xs$ *and* $\texttt{doEditsAnnot}(edits', xs') = ys'$ *(so that* $\texttt{annotListToList}\, ys' = ys$*), then* $\texttt{doEditsAnnot}(edits, xs') = ys'$*.*

Proof. Suppose xs and ys have type `'a list`, $edits$ has type `'a edit list`, $|edits| = 2$, the first element of $edits$ is an insert/append, the second element of $edits$ is a transpose, and $\texttt{doEdits}(edits, xs) = ys$. There are 5 cases to consider.

- Suppose $edits = [\texttt{Insert}(n, x); \texttt{Transpose}\, m]$ and $m < n - 1$. Then $edits' = [\texttt{Transpose}\, m; \texttt{Insert}(n, x)]$ has the property of (2).
- Suppose $edits = [\texttt{Insert}(n, x); \texttt{Transpose}(n - 1)]$. Then $edits' = [\texttt{Insert}(n - 1, x)]$ has the property of (1).
- Suppose $edits = [\texttt{Insert}(n, x); \texttt{Transpose}\, n]$. There are two subcases to consider.
  - Suppose $n < |xs| - 1$. Then $edits' = [\texttt{Insert}(n + 1, x)]$ has the property of (1).
  - Suppose $n = |xs| - 1$. Then $edits' = [\texttt{Append}\, x]$ has the property of (1).
- Suppose $edits = [\texttt{Insert}(n, x); \texttt{Transpose}\, m]$ and $m > n$. Then $edits' = [\texttt{Transpose}(m - 1); \texttt{Insert}(n, x)]$ has the property of (2).
- Suppose $edits = [\texttt{Append}\, x; \texttt{Transpose}\, n]$. Then there are two subcases to consider.
  - Suppose $n < |xs| - 1$. Then $edits' = [\texttt{Transpose}\, n; \texttt{Append}\, x]$ has the property of (2).
  - Suppose $n = |xs| - 1$. Then $edits' = [\texttt{Insert}(n, x)]$ has the property of (1).

□

Lemma 1.5
For all xs and ys of type `'a list` *and edits of type* `'a edit list`*, if* $\texttt{doEdits}(edits, xs) = ys$*, then either:*

(1) there is an $edits'$ *of type* `'a edit list` *such that* $|edits'| < |edits|$ *and* $\texttt{doEdits}(edits', xs) = ys$*; or*

(2) there is an $edits'$ *of type* `'a edit list` *such that* $|edits'| = |edits|$*,* $edits'$ *is standard,* $edits'$ *contains one or more deletes iff edits contains one or more deletes,* $\texttt{doEdits}(edits', xs) = ys$ *and, for all* xs'*,* ys' *of type* `'a annot list`*, if* $\texttt{annotListToList}\, xs' = xs$ *and* $\texttt{doEditsAnnot}(edits', xs') = ys'$ *(so that* $\texttt{annotListToList}\, ys' = ys$*), then* $\texttt{doEditsAnnot}(edits, xs') = ys'$*.*

Proof. Define a predicate $Q(edits, xs, ys)$ between *edits* of type $'a$ `edit list`, and xs and ys of type $'a$ `list`, by: $Q(edits, xs, ys)$ iff either:

(1) there is an $edits'$ of type $'a$ `edit list` such that $|edits'| < |edits|$ and $\texttt{doEdits}(edits', xs) = ys$; or

(2) there is an $edits'$ of type $'a$ `edit list` such that $|edits'| = |edits|$, $edits'$ is standard, $edits'$ contains one or more deletes iff *edits* contains one or more deletes, $\texttt{doEdits}(edits', xs) = ys$ and, for all xs', ys' of type $'a$ `annot list`, if $\texttt{annotListToList}\, xs' = xs$ and $\texttt{doEditsAnnot}(edits', xs') = ys'$ (so that $\texttt{annotListToList}\, ys' = ys$), then $\texttt{doEditsAnnot}(edits, xs') = ys'$.

We use complete induction on the length of the list of edits. Let $P(n)$ be the predicate on $\mathbb{N}$ defined by: $P(n)$ iff, for all *edits* of type $'a$ `edit list` and xs and ys of type $'a$ `list`, if $|edits| = n$ and $\texttt{doEdit}(edits, xs) = ys$, then $Q(edits, xs, ys)$. It will suffice to show that, for all $n \in \mathbb{N}$, $P(n)$, and we proceed by complete induction. Suppose $n \in \mathbb{N}$, and assume the inductive hypothesis, for all $m \in \mathbb{N}$, if $m < n$, then $P(m)$. We must show that $P(n)$.

First, we prove **Fact 1**: for all *edits* and $edits'$ of type $'a$ `edit list` and xs and ys of type $'a$ `list`, if $|edits'| = |edits|$, $edits'$ contains one or more deletes iff *edits* contains one or more deletes, for all xs' and ys' of type $'a$ `annot list`, if $\texttt{annotListToList}\, xs' = xs$ and $\texttt{doEditsAnnot}(edits', xs') = ys'$, then $\texttt{doEditsAnnot}(edits, xs') = ys'$, and $Q(edits', xs, ys)$, then $Q(edits, xs, ys)$. Suppose *edits* and $edits'$ have type $'a$ `edit list`, xs and ys have type $'a$ `list`, $|edits'| = |edits|$, $edits'$ contains one or more deletes iff *edits* contains one or more deletes, for all xs' and ys' of type $'a$ `annot list`, if $\texttt{annotListToList}\, xs' = xs$ and $\texttt{doEditsAnnot}(edits', xs') = ys'$, then $\texttt{doEditsAnnot}(edits, xs') = ys'$, and $Q(edits', xs, ys)$. We must show $Q(edits, xs, ys)$. Since $Q(edits', xs, ys)$, there are two cases to consider.

(1) Suppose there is an $edits''$ of type $'a$ `edit list` such that $|edits''| < |edits'|$ and $\texttt{doEdits}(edits'', xs) = ys$. Then we have that $|edits''| < |edits'| = |edits|$, so that $Q(edits, xs, ys)$.

(2) Suppose there is an $edits''$ of type $'a$ `edit list` such that $|edits''| = |edits'|$, $edits''$ is standard, $edits''$ contains one or more deletes iff $edits'$ contains one or more deletes, $\texttt{doEdits}(edits'', xs) = ys$ and, for all xs', ys' of type $'a$ `annot list`, if $\texttt{annotListToList}\, xs' = xs$ and $\texttt{doEditsAnnot}(edits'', xs') = ys'$, then $\texttt{doEditsAnnot}(edits', xs') = ys'$. Then $|edits''| = |edits|$, $edits''$ is standard, $edits''$ contains one or more deletes iff *edits* contains one or more deletes, $\texttt{doEdits}(edits'', xs) = ys$ and, and, for all xs', ys' of type $'a$ `annot list`, if $\texttt{annotListToList}\, xs' = xs$ and $\texttt{doEditsAnnot}(edits'', xs') = ys'$, then $\texttt{doEditsAnnot}(edits, xs') = ys'$, so that $Q(edits, xs, ys)$.

Next, we prove **Fact 2**: for all *edits* of type $'a$ `edit list` and xs and ys of type $'a$ `list`, if $|edits| = n$, $\texttt{doEdits}(edits, xs) = ys$ and *edits* begins with a delete, then $Q(edits, xs, ys)$. Suppose *edits* has type $'a$ `edit list`, xs and ys have type $'a$ `list`, $|edits| = n$, $\texttt{doEdits}(edits, xs) = ys$ and *edits* begins with a delete. We must show $Q(edits, xs, ys)$. Thus $edits = [edit]@edits_1$, for some *edit* of type $'a$ `edit` and $edits_1$ of type $'a$ `edit list`, where *edit* is a delete. Because $\texttt{doEdits}(edits, xs) = ys$, there is a zs of type $'a$ `list` such that $\texttt{doEdit}(edit, xs) = zs$ and $\texttt{doEdits}(edits_1, zs) = ys$. Since $|edits_1| < |edits| = n$, we have $P(|edits_1|)$. Thus $Q(edits_1, zs, ys)$, so that there are two cases to consider.

(1) Suppose there is an $edits_1'$ of type $'a$ `edit list` such that $|edits_1'| < |edits_1|$ and $\texttt{doEdits}(edits_1', zs) = ys$. Let $edits' = [edit]@edits_1'$. Then $|edits'| = |edits_1'|+1 < |edits_1|+1 = |edits|$ and $\texttt{doEdits}(edits', xs) = ys$, so that $Q(edits, xs, ys)$.

(2) Suppose there is an $edits_1'$ of type $'a$ `edit list` such that $|edits_1'| = |edits_1|$, $edits_1'$ is standard, $edits_1'$ contains one or more deletes iff $edits_1$ contains one or more deletes, $\texttt{doEdits}(edits_1', zs) = ys$ and, for all zs', ys' of type $'a$ `annot list`, if $\texttt{annotListToList}\, zs' = zs$ and $\texttt{doEditsAnnot}(edits_1', zs') = ys'$, then $\texttt{doEditsAnnot}(edits_1, zs') = ys'$. Let $edits' = [edit] \mathbin{@} edits_1'$. Then $|edits'| = |edits_1'| + 1 = |edits_1| + 1 = |edits|$, $edits'$ is standard, and $edits'$ contains one or more deletes iff $edits$ contains one or more deletes, and $\texttt{doEdits}(edits', xs) = ys$. To complete the proof that $Q(edits, xs, ys)$, suppose xs' and ys' have type $'a$ `annot list`, $\texttt{annotListToList}\, xs' = xs$ and $\texttt{doEditsAnnot}(edits', xs') = ys'$. Thus there is a zs' such that $\texttt{doEditAnnot}(edit, xs') = zs'$ and $\texttt{doEditsAnnot}(edits_1', zs') = ys'$, and hence $\texttt{annotListToList}\, zs' = zs$. Consequently, $\texttt{doEditsAnnot}(edits_1, zs') = ys'$, and thus $\texttt{doEditsAnnot}(edits, xs') = ys'$.

Next, we prove **Fact 3**: for all $edits$ of type $'a$ `edit list` and xs and ys of type $'a$ `list`, if $|edits| = n$, $\texttt{doEdits}(edits, xs) = ys$ and $edits$ consists of a transpose, followed by a list of edits with no deletes, then $Q(edits, xs, ys)$. Suppose $edits$ has type $'a$ `edit list`, xs and ys have type $'a$ `list`, $|edits| = n$, $\texttt{doEdits}(edits, xs) = ys$ and $edits$ consists of a transpose, followed by a list of edits with no deletes. We must show $Q(edits, xs, ys)$. Thus $edits = [edit] \mathbin{@} edits_1$, for some $edit$ of type $'a$ `edit` and $edits_1$ of type $'a$ `edit list`, where $edit$ is a transpose, and $edits_1$ has no deletes. Because $\texttt{doEdits}(edits, xs) = ys$, there is a zs of type $'a$ `list` such that $\texttt{doEdit}(edit, xs) = zs$ and $\texttt{doEdits}(edits_1, zs) = ys$. Since $|edits_1| < |edits| = n$, we have $P(|edits_1|)$. Thus $Q(edits_1, zs, ys)$, so that there are two cases to consider.

(1) Suppose there is an $edits_1'$ of type $'a$ `edit list` such that $|edits_1'| < |edits_1|$ and $\texttt{doEdits}(edits_1', zs) = ys$. Let $edits' = [edit]@edits_1'$. Then $|edits'| = |edits_1'|+1 < |edits_1|+1 = |edits|$ and $\texttt{doEdits}(edits', xs) = ys$, so that $Q(edits, xs, ys)$.

(2) Suppose there is an $edits_1'$ of type $'a$ `edit list` such that $|edits_1'| = |edits_1|$, $edits_1'$ is standard, $edits_1'$ contains one or more deletes iff $edits_1$ contains one or more deletes, $\texttt{doEdits}(edits_1', zs) = ys$ and, for all zs', ys' of type $'a$ `annot list`, if $\texttt{annotListToList}\, zs' = zs$ and $\texttt{doEditsAnnot}(edits_1', zs') = ys'$, then $\texttt{doEditsAnnot}(edits_1, zs') = ys'$. Since $edits_1$ has no deletes, it follows that $edits_1'$ has no deletes. Let $edits' = [edit] \mathbin{@} edits_1'$. Then $|edits'| = |edits_1'| + 1 = |edits_1| + 1 = |edits|$, $edits'$ is standard, $edits'$ contains one or more deletes iff $edits$ contains one or more deletes, and $\texttt{doEdits}(edits', xs) = ys$. To complete the proof that $Q(edits, xs, ys)$, suppose xs' and ys' have type $'a$ `annot list`, $\texttt{annotListToList}\, xs' = xs$ and $\texttt{doEditsAnnot}(edits', xs') = ys'$. Thus there is a zs' such that $\texttt{doEditAnnot}(edit, xs') = zs'$ and $\texttt{doEditsAnnot}(edits_1', zs') = ys'$, and hence $\texttt{annotListToList}\, zs' = zs$. Consequently, $\texttt{doEditsAnnot}(edits_1, zs') = ys'$, and thus $\texttt{doEditsAnnot}(edits, xs') = ys'$.

Now, we use our facts to prove $P(n)$. Suppose $edits$ has type $'a$ `edit list`, xs and ys have type $'a$ `list`, $|edits| = n$ and $\texttt{doEdit}(edits, xs) = ys$. We must show $Q(edits, xs, ys)$.

If $edits = [\,]$, then we can let $edits' = [\,]$. Then $|edits'| = |edits|$, $edits'$ is standard, $edits'$ contains one or more deletes iff $edits$ contains one or more deletes, $\texttt{doEdits}(edits', xs) = ys$ and, for all xs',

ys' of type $'$`a annot list`, if `annotListToList` $xs' = xs$ and `doEditsAnnot`$(edits', xs') = ys'$, then `doEditsAnnot`$(edits, xs') = ys'$. Thus $Q(edits, xs, ys)$.

So, suppose $edits = [edit_1] @ edits_1$ for some $edit_1$ of type $'$`a edit` and $edits_1$ of type $'$`a edit list`. Because `doEdits`$(edits, xs) = ys$, there is a zs of type $'$`a list` such that `doEdit`$(xs, edit_1) = zs$ and `doEdits`$(edits_1, zs) = ys$. Because $|edits_1| < |edits| = n$, $P(|edits_1|)$ holds. The $Q(edits_1, zs, ys)$, so there are two cases to consider.

(1) Suppose there is an $edits'_1$ of type $'$`a edit list` such that $|edits'_1| < |edits_1|$ and `doEdits`$(edits'_1, zs) = ys$. Let $edits' = edit_1 @ edits'_1$. Then $|edits'| = 1 + |edits'_1| < 1 + |edits_1| = |edits|$ and `doEdits`$(edits', xs) = ys$. Thus $Q(edits, xs, ys)$.

(2) Suppose there is an $edits'_1$ of type $'$`a edit list` such that $|edits'_1| = |edits_1|$, $edits'_1$ is standard, $edits'_1$ contains one or more deletes iff $edits_1$ contains one or more deletes, `doEdits`$(edits'_1, zs) = ys$ and, for all zs', ys' of type $'$`a annot list`, if `annotListToList` $zs' = zs$ and `doEditsAnnot`$(edits'_1, zs') = ys'$, then `doEditsAnnot`$(edits_1, zs') = ys'$.

If $[edit_1] @ edits'_1$ is standard, then let $edits' = [edit_1] @ edits'_1$. Then $|edits'| = |edits|$, $edits'$ is standard, $edits'$ contains one or more deletes iff $edits$ contains one or more deletes, `doEdits`$(edits', xs) = ys$ and, for all xs', ys' of type $'$`a annot list`, if `annotListToList` $xs' = xs$ and `doEditsAnnot`$(edits', xs') = ys'$, then `doEditsAnnot`$(edits, xs') = ys'$. (For the last of these, suppose xs' and ys' have type $'$`a annot list`, `annotListToList` $xs' = xs$ and `doEditsAnnot`$(edits', xs') = ys'$. Then there is a zs' of type $'$`a annot list` such that `doEdit`$(edit_1, xs') = zs'$, `doEditsAnnot`$(edits'_1, zs') = ys'$ and `annotListToList` $zs' = zs$. Hence `doEditsAnnot`$(edits_1, zs') = ys'$, so that `doEditsAnnot`$(edits', xs') = ys'$.) Thus $Q(edits, xs, ys)$.

Otherwise, $edit_1$ is not a delete and $edits'_1$ doesn't begin with an insert/append, and thus there are two cases to consider.

(A) Suppose $edits'_1$ begins with a delete, so that $edits'_1 = [edit'_2] @ edits'_2$ and $edit'_2$ is a delete. Because `doEdits`$(edits'_1, zs) = ys$, there is a ws of type $'$`a list` such that `doEdit`$(edit'_2, zs) = ws$ and `doEdits`$(edits'_2, ws) = ys$. Because `doEdits`$([edit_1; edit'_2], xs) = ws$, $edit_1$ is not a delete, and $edit'_2$ is a delete, we can apply Lemma 1.3, giving us two cases to consider.

(a) Suppose there is an $edits''$ of type $'$`a edit list` such that $|edits''| < 2$ and `doEdits`$(edits'', xs) = ws$. Let $edits' = edits'' @ edits'_2$. Then $|edits'| < |edits|$ and `doEdits`$(edits', xs) = ys$. Hence $Q(edits, xs, ys)$.

(b) Suppose there is an $edits''$ of type $'$`a edit list` such that $|edits''| = 2$, the first element of $edits''$ is a delete, the second element of $edits''$ is not a delete, `doEdits`$(edits'', xs) = ws$, and, for all xs' and ws' of type $'$`a annot list`, if `annotListToList` $xs' = xs$ and `doEditsAnnot`$(edits'', xs') = ws'$, then `doEditsAnnot`$([edit_1; edit'_2], xs') = ws'$. Let $edits' = edits'' @ edits'_2$. Then $|edits'| = n = |edits|$, `doEdits`$(edits', xs) = ys$, and $edits'$ begins with a delete, so that $Q(edits', xs, ys)$, by **Fact 2**. Because $edits'_1$ contains one or more deletes, so does $edits_1$, and thus $edits'$ contains one or more deletes iff $edits$ contains one or more deletes. Thus, by **Fact 1**, to conclude $Q(edits, xs, ys)$, it will suffice to

show that, for all xs' and ys' of type $'$`a annot list`, if `annotListToList` $xs' = xs$ and `doEditsAnnot`$(edits', xs') = ys'$, then `doEditsAnnot`$(edits, xs') = ys'$. Suppose xs' and ys' have type $'$`a annot list`, `annotListToList` $xs' = xs$ and `doEditsAnnot`$(edits', xs') = ys'$. Thus there is a ws' of type $'$`a annot list` such that `doEditsAnnot`$(edits'', xs') = ws'$, `doEditsAnnot`$(edits'_2, ws') = ys'$, and `annotListToList` $ws' = ws$. Thus `doEditsAnnot`$([edit_1; edit'_2], xs') = ws'$, so that there is a zs' of type $'$`a annot list` such that `doEditAnnot`$(edit_1, xs') = zs'$, `doEditAnnot`$(edit'_2, zs') = ws'$ and `annotListToList` $zs' = zs$. Hence we have `doEditsAnnot`$(edits'_1, zs') = ys'$, so that `doEditsAnnot`$(edits_1, zs') = ys'$. Finally, `doEditsAnnot`$(edits, xs') = ys'$.

(B) Suppose $edits'_1$ begins with a transpose, so that $edit_1$ is an insert/append. Thus $edits'_1 = [edit'_2]$ @ $edits'_2$ and $edit'_2$ is a transpose. Since $edits'_1$ is standard, it contains no deletes. Because `doEdits`$(edits'_1, zs) = ys$, there is a ws of type $'$`a list` such that `doEdit`$(edit'_2, zs) = ws$ and `doEdits`$(edits'_2, ws) = ys$. Because `doEdits`$([edit_1; edit'_2], xs) = ws$, $edit_1$ is an insert/append, and $edit'_2$ is a transpose, we can apply Lemma 1.4, giving us two cases to consider.

(a) Suppose there is an $edits''$ of type $'$`a edit list` such that $|edits''| < 2$ and `doEdits`$(edits'', xs) = ws$. Let $edits' = edits''$ @ $edits'_2$. Then $|edits'| < |edits|$ and `doEdits`$(edits', xs) = ys$. Hence $Q(edits, xs, ys)$.

(b) Suppose there is an $edits''$ of type $'$`a edit list` such that $|edits''| = 2$, the first element of $edits''$ is a transpose, the second element of $edits''$ is an insert/append, `doEdits`$(edits'', xs) = ws$, and, for all xs' and ws' of type $'$`a annot list`, if `annotListToList` $xs' = xs$ and `doEditsAnnot`$(edits'', xs') = ws'$, then `doEditsAnnot`$([edit_1; edit'_2], xs') = ws'$. Let $edits' = edits''$ @ $edits'_2$. Then $|edits'| = n = |edits|$, `doEdits`$(edits', xs) = ys$, and $edits'$ consists of a transpose, followed by a list of edits with no deletes, so that $Q(edits', xs, ys)$, by **Fact 3**. Because $edits'_1$ contains no deletes, neither does $edits_1$, and thus $edits'$ contains one or more deletes iff $edits$ contains one or more deletes. Thus, by **Fact 1**, to conclude $Q(edits, xs, ys)$, it will suffice to show that, for all xs' and ys' of type $'$`a annot list`, if `annotListToList` $xs' = xs$ and `doEditsAnnot`$(edits', xs') = ys'$, then `doEditsAnnot`$(edits, xs') = ys'$, and this follows as in the preceding case.

□

Now, we prove the Fundamental Lemma for Annotated Lists. Suppose xs and ys are of type $'$`a list`. We use complete induction on the length of the list of edits. Let $P(n)$ be the predicate on $\mathbb{N}$ defined by: $P(n)$ iff, for all $edits$ of type $'$`a edit list`, if $|edits| = n$, `doEdit`$(edits, xs) = ys$ and `doEditsAnnot`$(edits,$ `listToAnnotList` $xs)$ raises `Error`, then there is an $edits'$ of type $'$`a edit list` such that $|edits'| < |edits|$ and `annotListToList`(`doEditsAnnot`$(edits',$ `listToAnnotList` $xs)) = ys$. It will suffice to show that, for all $n \in \mathbb{N}$, $P(n)$, and we proceed by complete induction.

Suppose $n \in \mathbb{N}$, and assume the inductive hypothesis, for all $m \in \mathbb{N}$, if $m < n$, then $P(m)$. We must show that $P(n)$. Suppose $edits$ has type $'$`a edit list`, $|edits| = n$, `doEdit`$(edits, xs) = ys$ and `doEditsAnnot`$(edits,$ `listToAnnotList` $xs)$ raises `Error`. We

must show that there is an $edits'$ of type $'\texttt{a edit list}$ such that $|edits'| < |edits|$ and $\texttt{annotListToList}(\texttt{doEditsAnnot}(edits', \texttt{listToAnnotList}\, xs)) = ys$.

By Lemma 1.5, there are two cases to consider:

(1) Suppose there is an $edits'$ of type $'\texttt{a edit list}$ such that $|edits'| < |edits|$ and $\texttt{doEdits}(edits', xs) = ys$. If $\texttt{annotListToList}(\texttt{doEditsAnnot}(edits', \texttt{listToAnnotList}\, xs)) = ys$, then we are done. Otherwise, $\texttt{doEditsAnnot}(edits', \texttt{listToAnnotList}\, xs)$ raises `Error`. Because $|edits'| < |edits| = n$, we have $P(|edits'|)$. Thus there is an $edits''$ of type $'\texttt{a edit list}$ such that $|edits''| < |edits'|$ and $\texttt{annotListToList}(\texttt{doEditsAnnot}(edits'', \texttt{listToAnnotList}\, xs)) = ys$. And, since $|edits''| < |edits'| < |edits|$, we have that that $|edits''| < |edits|$.

(2) Suppose there is an $edits'$ of type $'\texttt{a edit list}$ such that $|edits'| = |edits|$, $edits'$ is standard, $\texttt{doEdits}(edits', xs) = ys$ and, for all xs', ys' of type $'\texttt{a annot list}$, if $\texttt{annotListToList}\, xs' = xs$ and $\texttt{doEditsAnnot}(edits', xs') = ys'$ (so that $\texttt{annotListToList}\, ys' = ys$), then $\texttt{doEditsAnnot}(edits, xs') = ys'$. Because $edits'$ is standard and $\texttt{doEdits}(edits', xs) = ys$, Lemma 1.2 tells us that $\texttt{annotListToList}(\texttt{doEditsAnnot}(edits', \texttt{listToAnnotList}\, xs)) = ys$. Define xs' and ys' of type $'\texttt{a annot list}$ by: $xs' = \texttt{listToAnnotList}\, xs$ and $ys' = \texttt{doEditsAnnot}(edits', xs')$. Since $\texttt{annotListToList}\, xs' = xs$ and $\texttt{doEditsAnnot}(edits', xs') = ys'$, we have that $\texttt{doEditsAnnot}(edits, \texttt{listToAnnotList}\, xs) = \texttt{doEditsAnnot}(edits, xs') = ys'$—contradiction. Thus there is an $edits'$ of type $'\texttt{a edit list}$ such that $|edits'| < |edits|$ and $\texttt{annotListToList}(\texttt{doEditsAnnot}(edits', \texttt{listToAnnotList}\, xs)) = ys$.

Exercise 2

Lemma 2.1
Suppose xs and ys have type $'\texttt{a list}$, $bound \in \mathbb{N}$, $edits$ has type $'\texttt{a edit list}$, $state$ has type $'\texttt{a state}$, ans has type $'\texttt{a edit list list}$, $\texttt{consistent}(state, xs, ys, edits)$, $\texttt{not}(\texttt{success}(state, ys))$, and f is the function of type

$$'\texttt{a choice} * '\texttt{a edit list list} \to '\texttt{a edit list list}$$

such that, for all valid choices $(edit, state')$ for $state$ and ans of type $'\texttt{a edit list list}$, $f((edit, state'), ans)$ returns $ans' \mathbin{@} ans$, where ans' is the list of all edit-lists $edits'$ such that

- $\texttt{prefix}(edits \mathbin{@} [edit], edits')$,
- $\texttt{annotListToList}(\texttt{doEditsAnnot}(edits', \texttt{listToAnnotList}\, xs)) = ys$,
- $|edits'| \leq bound$ *and*
- $\texttt{noProperPrefixSucceeds}(edits', xs, ys)$,

listed in strictly ascending order. Then $\texttt{choices}(state, f, ans)$ returns $ans' \mathbin{@} ans$, where ans' is the list of all edit-lists $edits'$ such that

- $\texttt{prefix}(edits, edits')$,
- $\texttt{annotListToList}(\texttt{doEditsAnnot}(edits', \texttt{listToAnnotList}\, xs)) = ys$,

- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`$(edits', xs, ys)$,

listed in strictly ascending order.

Proof. Because `consistent`$(state, xs, ys, edits)$ and `not`(`success`$(state, ys)$), we have that `annotListToList`(`doEditsAnnot`$(edits,$ `listToAnnotList` $xs)) \neq ys$. And `consistent`$(state, xs, ys,$ $edits)$ tells us that `valid`$(state)$.

Let zs of type $'a$ `choice list` be the choices that are valid for $state$, listed in order without duplicates. Then `choices`$(state, f, ans)$ returns `fold`(zs, f, ans). Thus we must show that `fold`(zs, f, ans) returns $ans' \mathbin{@} ans$, where ans' is the list of all edit-lists $edits'$ such that

- `prefix`$(edits, edits')$,
- `annotListToList`(`doEditsAnnot`$(edits',$ `listToAnnotList` $xs)) = ys$,
- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`$(edits', xs, ys)$,

listed in strictly ascending order.

First, we prove that for all us of type $'a$ `choice list`, if us is a suffix (trailing part) of zs, then `fold`(us, f, ans) returns $ans' \mathbin{@} ans$, where ans' is the list of all edit-lists $edits'$ such that

- `prefix`$(edits \mathbin{@} [edit], edits')$, for some $edit$ of type $'a$ `edit` that appears on the left side of an element of us,
- `annotListToList`(`doEditsAnnot`$(edits',$ `listToAnnotList` $xs)) = ys$,
- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`$(edits', xs, ys)$,

listed in strictly ascending order. We proceed by induction on us.

(Basis Step) We must prove that, if $[\,]$ is a suffix of zs, then `fold`$([\,], f, ans)$ returns $ans' \mathbin{@} ans$, where ans' is the list of all edit-lists $edits'$ such that

- `prefix`$(edits \mathbin{@} [edit], edits')$, for some $edit$ of type $'a$ `edit` that appears on the left side of an element of $[\,]$,
- `annotListToList`(`doEditsAnnot`$(edits',$ `listToAnnotList` $xs)) = ys$,
- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`$(edits', xs, ys)$,

listed in strictly ascending order. Suppose $[\,]$ is a prefix of zs (which always holds). But `fold`$([\,], f, ans) = ans$ and thus we must show that there is no edit-list $edits'$ such that

- `prefix`$(edits \mathbin{@} [edit], edits')$, for some $edit$ of type $'a$ `edit` that appears on the left side of an element of $[\,]$,

- `annotListToList`(`doEditsAnnot`($edits'$, `listToAnnotList` xs)) = ys,
- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`($edits'$, xs, ys).

This follows because the first part is impossible.

(Inductive Step) Suppose us has type `'a choice list`, $(edit, state')$ has type `'a choice`, and assume the inductive hypothesis, if us is a suffix of zs, then `fold`(us, f, ans) returns $ans' \mathbin{@} ans$, where ans' is the list of all edit-lists $edits'$ such that

- `prefix`($edits \mathbin{@} [edit']$, $edits'$), for some $edit'$ of type `'a edit` that appears on the left side of an element of us,
- `annotListToList`(`doEditsAnnot`($edits'$, `listToAnnotList` xs)) = ys,
- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`($edits'$, xs, ys),

listed in strictly ascending order. We must prove that, if $[(edit, state')] \mathbin{@} us$ is a suffix of zs, then `fold`($[(edit, state')] \mathbin{@} us$, f, ans) returns $ans' \mathbin{@} ans$, where ans' is the list of all edit-lists $edits'$ such that

- `prefix`($edits \mathbin{@} [edit']$, $edits'$), for some $edit'$ of type `'a edit` that appears on the left side of an element of $[(edit, state')] \mathbin{@} us$,
- `annotListToList`(`doEditsAnnot`($edits'$, `listToAnnotList` xs)) = ys,
- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`($edits'$, xs, ys),

listed in strictly ascending order. Suppose $[(edit, state')] \mathbin{@} us$ is a suffix of zs. Thus $(edit, state')$ is a valid choice for $state$, and us is a suffix of zs, so that the inductive hypothesis tells us that `fold`(us, f, ans) returns $ans' \mathbin{@} ans$, where ans' is the list of all edit-lists $edits'$ such that

- `prefix`($edits \mathbin{@} [edit']$, $edits'$), for some $edit'$ of type `'a edit` that appears on the left side of an element of us,
- `annotListToList`(`doEditsAnnot`($edits'$, `listToAnnotList` xs)) = ys,
- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`($edits'$, xs, ys),

listed in strictly ascending order. We have that

$$\begin{aligned} \texttt{fold}([(edit, state')] \mathbin{@} us, f, ans) &= f((edit, state'), \texttt{fold}(us, f, ans)) \\ &= f((edit, state'), ans' \mathbin{@} ans) \\ &= ans'' \mathbin{@} ans' \mathbin{@} ans, \end{aligned}$$

where ans'' is the list of all edit-lists $edits'$ such that

- `prefix`($edits \mathbin{@} [edit]$, $edits'$),

- `annotListToList`(`doEditsAnnot`($edits'$, `listToAnnotList` xs)) = ys,
- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`($edits'$, xs, ys),

listed in strictly ascending order. So we must show that $ans'' \mathbin{@} ans'$ consists of all edit-lists $edits'$ such that

- `prefix`($edits \mathbin{@} [edit']$, $edits'$), for some $edit'$ of type $'$`a edit` that appears on the left side of an element of $[(edit, state')] \mathbin{@} us$,
- `annotListToList`(`doEditsAnnot`($edits'$, `listToAnnotList` xs)) = ys,
- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`($edits'$, xs, ys),

listed in strictly ascending order.

We know that $[(edit, state')] \mathbin{@} us$ is sorted and has no duplicates. Thus Valid Choice Lemma 0 tells us that no element of us has $edit$ as its left side. Hence the left sides of the elements in us are all strictly greater than $edit$. Consequently, $ans'' \mathbin{@} ans'$ is sorted and contains no duplicates.

So it remains to show that an $edits'$ is in $ans'' \mathbin{@} ans'$ iff

- `prefix`($edits \mathbin{@} [edit']$, $edits'$), for some $edit'$ of type $'$`a edit` that appears on the left side of an element of $[(edit, state')] \mathbin{@} us$,
- `annotListToList`(`doEditsAnnot`($edits'$, `listToAnnotList` xs)) = ys,
- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`($edits'$, xs, ys),

which is obvious.

By the preceding induction, we have that `fold`(zs, f, ans) returns $ans' \mathbin{@} ans$, where ans' is the list of all edit-lists $edits'$ such that

- `prefix`($edits \mathbin{@} [edit]$, $edits'$), for some $edit$ of type $'$`a edit` that appears on the left side of an element of zs, i.e., on the left side of a valid choice for $state$,
- `annotListToList`(`doEditsAnnot`($edits'$, `listToAnnotList` xs)) = ys,
- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`($edits'$, xs, ys),

listed in strictly ascending order.

Thus it remains to show that, for all edit-lists $edits'$:

- `prefix`($edits$, $edits'$),
- `annotListToList`(`doEditsAnnot`($edits'$, `listToAnnotList` xs)) = ys,

- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`$(edits', xs, ys)$,

iff

- `prefix`$(edits \mathbin{@} [edit], edits')$, for some *edit* of type `'a edit` that appears on the left side of a valid choice for *state*,
- `annotListToList`$($`doEditsAnnot`$(edits',$ `listToAnnotList` $xs)) = ys$,
- $|edits'| \leq bound$ and
- `noProperPrefixSucceeds`$(edits', xs, ys)$,

The "if" (bottom-to-top) direction is obvious. For the "only if" (top-to-bottom) direction, since `annotListToList`$($`doEditsAnnot`$(edits,$ `listToAnnotList` $xs)) \neq ys$, we have that $edits' \neq edits$. Thus Valid Choice Lemma 2 tells us that there are *edit*, $edits''$ and $state'$ such that $edits' = edits \mathbin{@} [edit] \mathbin{@} edits''$ and $(edit, state')$ is a valid choice for *state*. Thus the bottom condition holds. □

Suppose *bound* is an integer, $bound \geq 0$ and *xs* and *ys* have type `'a edit list`. We must show that `boundedFind`$(bound, xs, ys)$ returns the list of all edit-lists *edits* such that

- `annotListToList`$($`doEditsAnnot`$(edits,$ `listToAnnotList` $xs)) = ys$,
- $|edits| \leq bound$ and
- `noProperPrefixSucceeds`$(edits, xs, ys)$,

listed in strictly ascending order.

First, we must show that the auxiliary function `find` is correct.

Lemma 2.2
For all $l \in \mathbb{N}$, for all editsRev of type `'a edit list`*, integers len, state of type* `'a state` *and ans of type* `'a edit list list`*, if $bound - len = l$, then, if*

- $len \leq bound$,
- $len = |editsRev|$,
- `noProperPrefixSucceeds`$($`rev` $editsRev, xs, ys)$ *and*
- `consistent`$(state, xs, ys,$ `rev` $editsRev)$,

then `find`$(editsRev, len, state, ans)$ *returns $ans'@ans$, where ans' is the list of all edit-lists edits such that*

- `prefix`$($`rev` $editsRev, edits)$,
- `annotListToList`$($`doEditsAnnot`$(edits,$ `listToAnnotList` $xs)) = ys$,
- $|edits| \leq bound$ *and*

- `noProperPrefixSucceeds`($edits, xs, ys$),

listed in strictly ascending order.

Proof. Define a predicate $P(l)$ on natural numbers by: $P(l)$ iff for all *editsRev* of type `'a edit list`, integers *len*, *state* of type `'a state` and *ans* of type `'a edit list list`, if $bound - len = l$, then, if

- $len \leq bound$,
- $len = |editsRev|$,
- `noProperPrefixSucceeds`(`rev` *editsRev*, xs, ys) and
- `consistent`($state, xs, ys$, `rev` *editsRev*),

then `find`(*editsRev*, *len*, *state*, *ans*) returns $ans'@ans$, where ans' is the list of all edit-lists *edits* such that

- `prefix`(`rev` *editsRev*, *edits*),
- `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` xs)) $= ys$,
- $|edits| \leq bound$ and
- `noProperPrefixSucceeds`($edits, xs, ys$),

listed in strictly ascending order. We use ordinary induction to prove that, for all $l \in \mathbb{N}$, $P(l)$.

(Basis Step) We must show $P(0)$. Suppose *editsRev* has type `'a edit list`, *len* is an integer, *state* has type `'a state`, *ans* has type `'a edit list list`, $bound - len = 0$ and

- $len \leq bound$,
- $len = |editsRev|$,
- `noProperPrefixSucceeds`(`rev` *editsRev*, xs, ys) and
- `consistent`($state, xs, ys$, `rev` *editsRev*).

We must show that `find`(*editsRev*, *len*, *state*, *ans*) returns $ans'@ans$, where ans' is the list of all edit-lists *edits* such that

- `prefix`(`rev` *editsRev*, *edits*),
- `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` xs)) $= ys$,
- $|edits| \leq bound$ and
- `noProperPrefixSucceeds`($edits, xs, ys$),

listed in strictly ascending order. Because $bound - len = 0$, it follows that $bound = len = |editsRev|$. There are three cases to consider.

- Suppose `impossible`($state, bound - len$), i.e., `impossible`($state, 0$). Then we have that `find`(*editsRev*, *len*, *state*, *ans*) returns *ans*, and so we must show that there are no edit-lists *edits* such that

  - `prefix`(`rev` *editsRev*, *edits*),
  - `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys*,
  - |*edits*| ≤ *bound* and
  - `noProperPrefixSucceeds`(*edits*, *xs*, *ys*),

  Suppose, toward a contradiction, that there is such an edit-list *edits*. Because `prefix`(`rev` *editsRev*, *edits*) and |*edits*| ≤ *bound*, we have that *edits* = `rev` *editsRev*. But, combining `consistent`(*state*, *xs*, *ys*, `rev` *editsRev*) and `impossible`(*state*, 0), we have that `annotListToList`(`doEditsAnnot`(`rev` *editsRev*, `listToAnnotList` *xs*)) = *ys* is false. Thus `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys* is false—contradiction. Thus we have our result.

- Suppose `impossible`(*state*, *bound* − *len*) is false, but `success`(*state*, *ys*). Then `find`(*editsRev*, *len*, *state*, *ans*) returns [`rev` *editsRev*] @ *ans*, and so we must show that `rev` *editsRev* is the unique edit-list *edits* such that
  - `prefix`(`rev` *editsRev*, *edits*),
  - `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys*,
  - |*edits*| ≤ *bound* and
  - `noProperPrefixSucceeds`(*edits*, *xs*, *ys*).

  We have that `prefix`(`rev` *editsRev*, `rev` *editsRev*). Furthermore, the combination of `consistent`(*state*, *xs*, *ys*, `rev` *editsRev*) and `success`(*state*, *ys*), lets us conclude that `annotListToList`(`doEditsAnnot`(`rev` *editsRev*, `listToAnnotList` *xs*)) = *ys*. We have that |*editsRev*| = *bound*, and thus |`rev` *editsRev*| ≤ *bound*. And we know `noProperPrefixSucceeds`(`rev` *editsRev*, *xs*, *ys*).

  Now, suppose that *edits* is an edit-list such that
  - `prefix`(`rev` *editsRev*, *edits*),
  - `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys*,
  - |*edits*| ≤ *bound* and
  - `noProperPrefixSucceeds`(*edits*, *xs*, *ys*),

  Since |*editsRev*| = *bound* and `prefix`(`rev` *editsRev*, *edits*), it follows that *edits* = `rev` *editsRev*, as required.

- Suppose `impossible`(*state*, *bound* − *len*) is false and `success`(*state*, *ys*) is false. Then `find`(*editsRev*, *len*, *state*, *ans*) returns *ans*, and so we must show that there are no edit-lists *edits* such that
  - `prefix`(`rev` *editsRev*, *edits*),
  - `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys*,
  - |*edits*| ≤ *bound* and
  - `noProperPrefixSucceeds`(*edits*, *xs*, *ys*),

  Suppose, toward a contradiction, that there is such an edit-list *edits*. Because `prefix`(`rev` *editsRev*, *edits*) and |*edits*| ≤ *bound*, we have that *edits* = `rev` *editsRev*. But, combining `consistent`(*state*, *xs*, *ys*, `rev` *editsRev*) and `not`(`success`(*state*, *ys*)), we have that `annotListToList`(`doEditsAnnot`(`rev` *editsRev*, `listToAnnotList` *xs*)) = *ys* is

false. Thus `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys* is false—contradiction. Thus we have our result.

(Inductive Step) Suppose $l \in \mathbb{N}$, and assume the inductive hypothesis, $P(l)$. We must show that $P(l+1)$. Suppose *editsRev* has type `'a edit list`, *len* is an integer, *state* has type `'a state`, *ans* has type `'a edit list list`, $bound - len = l + 1$ and

- $len \leq bound$,
- $len = |editsRev|$,
- `noProperPrefixSucceeds`(`rev` *editsRev*, *xs*, *ys*) and
- `consistent`(*state*, *xs*, *ys*, `rev` *editsRev*).

We must show that `find`(*editsRev*, *len*, *state*, *ans*) returns *ans*′@*ans*, where *ans*′ is the list of all edit-lists *edits* such that

- `prefix`(`rev` *editsRev*, *edits*),
- `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys*,
- $|edits| \leq bound$ and
- `noProperPrefixSucceeds`(*edits*, *xs*, *ys*),

listed in strictly ascending order. Because $0 \leq len \leq bound$ and $bound - len = l + 1$, it follows that $len < bound$. There are three cases to consider.

- Suppose `impossible`(*state*, $bound - len$). Then `find`(*editsRev*, *len*, *state*, *ans*) returns *ans*, and so we must show that there are no edit-lists *edits* such that
  - `prefix`(`rev` *editsRev*, *edits*),
  - `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys*,
  - $|edits| \leq bound$ and
  - `noProperPrefixSucceeds`(*edits*, *xs*, *ys*).

  Suppose, toward a contradiction, that there is such an edit-list *edits*. Because `prefix`(`rev` *editsRev*, *edits*) and $|edits| \leq bound$, we have that *edits* = `rev` *editsRev* @ *edits*′, for some *edits*′ of length no more than $bound - len$. But, combining `consistent`(*state*, *xs*, *ys*, `rev` *editsRev*) and `impossible`(*state*, $bound - len$), we have that `annotListToList`(`doEditsAnnot`(`rev` *editsRev* @ *edits*′, `listToAnnotList` *xs*)) = *ys* is false. Thus `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys* is false—contradiction. Thus we have our result.
- Suppose `impossible`(*state*, $bound - len$) is false, but `success`(*state*, *ys*). Then `find`(*editsRev*, *len*, *state*, *ans*) returns [`rev` *editsRev*] @ *ans*, and so we must show that `rev` *editsRev* is the unique edit-list *edits* such that
  - `prefix`(`rev` *editsRev*, *edits*),
  - `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys*,
  - $|edits| \leq bound$ and
  - `noProperPrefixSucceeds`(*edits*, *xs*, *ys*).

We have that `prefix`(`rev` *editsRev*, `rev` *editsRev*). Furthermore, putting together `consistent`(*state*, *xs*, *ys*, `rev` *editsRev*) and `success`(*state*, *ys*), we are able to conclude that `annotListToList`(`doEditsAnnot`(`rev` *editsRev*, `listToAnnotList` *xs*)) = *ys*. We have that |`rev` *editsRev*| = *len* ≤ *bound*. And finally, we know that `noProperPrefixSucceeds`(`rev` *editsRev*, *xs*, *ys*).

Now, suppose that *edits* is an edit-list such that

- `prefix`(`rev` *editsRev*, *edits*),
- `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys*,
- |*edits*| ≤ *bound* and
- `noProperPrefixSucceeds`(*edits*, *xs*, *ys*).

Suppose, toward a contradiction, that *edits* ≠ `rev` *editsRev*. Then, because `prefix`(`rev` *editsRev*, *edits*), we have that *edits* = `rev` *editsRev* @ *edits*′ for some nonempty edits-list *edits*′. But

$$\texttt{annotListToList}(\texttt{doEditsAnnot}(\texttt{rev}\ \mathit{editsRev}, \texttt{listToAnnotList}\ \mathit{xs})) = \mathit{ys},$$

and thus `noProperPrefixSucceeds`(*edits*, *xs*, *ys*) is false—contradiction. Thus we have our result.

- Suppose `impossible`(*state*, *bound* − *len*) is false and `success`(*state*, *ys*) is false. Then `find`(*editsRev*, *len*, *state*, *ans*) simplifies to `choices`(*state*, *f*, *ans*), where *f* is

$$\texttt{function}\ ((\mathit{edit}, \mathit{state}), \mathit{ans}) \rightarrow \texttt{find}([\mathit{edit}] \mathbin{@} \mathit{editsRev}, \mathit{len} + 1, \mathit{state}, \mathit{ans}).$$

We must show that `choices`(*state*, *f*, *ans*) returns *ans*′@*ans*, where *ans*′ is the list of all edit-lists *edits* such that

- `prefix`(`rev` *editsRev*, *edits*),
- `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys*,
- |*edits*| ≤ *bound* and
- `noProperPrefixSucceeds`(*edits*, *xs*, *ys*),

listed in strictly ascending order.

Next, we prove that, for all valid choices (*edit*, *state*′) for *state* and *ans* of type ′`a edit list list`, *f* ((*edit*, *state*′), *ans*) returns *ans*′ @ *ans*, where *ans*′ is the list of all edit-lists *edits* such that

- `prefix`(`rev` *editsRev* @ [*edit*], *edits*),
- `annotListToList`(`doEditsAnnot`(*edits*, `listToAnnotList` *xs*)) = *ys*,
- |*edits*| ≤ *bound* and
- `noProperPrefixSucceeds`(*edits*, *xs*, *ys*),

listed in strictly ascending order. Suppose (*edit*, *state*′) is a valid choice for *state* and *ans* has type ′`a edit list list`. By the inductive hypothesis, *P*(*l*), we have that, if *bound* − (*len* + 1) = *l*, then, if

- *len* + 1 ≤ *bound*,

- $len + 1 = |[edit] \mathbin{@} editsRev|$,
- $\texttt{noProperPrefixSucceeds}(\texttt{rev}([edit] \mathbin{@} editsRev), xs, ys)$ and
- $\texttt{consistent}(state', xs, ys, \texttt{rev}([edit] \mathbin{@} editsRev))$,

then $\texttt{find}([edit] \mathbin{@} editsRev, len + 1, state', ans)$ returns $ans'@ans$, where ans' is the list of all edit-lists *edits* such that

- $\texttt{prefix}(\texttt{rev}([edit] \mathbin{@} editsRev), edits)$,
- $\texttt{annotListToList}(\texttt{doEditsAnnot}(edits, \texttt{listToAnnotList}\, xs)) = ys$,
- $|edits| \leq bound$ and
- $\texttt{noProperPrefixSucceeds}(edits, xs, ys)$,

listed in strictly ascending order. Because $bound - (len+1) = l$ and $\texttt{rev}\, editsRev @ [edit] = \texttt{rev}([edit] \mathbin{@} editsRev)$, it will suffice to show that

- $len + 1 \leq bound$,
- $len + 1 = |[edit] \mathbin{@} editsRev|$,
- $\texttt{noProperPrefixSucceeds}(\texttt{rev}([edit] \mathbin{@} editsRev), xs, ys)$ and
- $\texttt{consistent}(state', xs, ys, \texttt{rev}([edit] \mathbin{@} editsRev))$.

Because $len < bound$, we have that $len + 1 \leq bound$. Since $len = |editsRev|$, we have that $len + 1 = |[edit] \mathbin{@} editsRev|$. Since $\texttt{noProperPrefixSucceeds}(\texttt{rev}\, editsRev, xs, ys)$ and $\texttt{annotListToList}(\texttt{doEditsAnnot}(\texttt{rev}\, editsRev, \texttt{listToAnnotList}\, xs)) \neq ys$, it follows that $\texttt{noProperPrefixSucceeds}(\texttt{rev}\, editsRev \mathbin{@} [edit], xs, ys)$, Consequently, $\texttt{noProperPrefixSucceeds}(\texttt{rev}([edit] \mathbin{@} editsRev), xs, ys)$. Finally, by Valid Choice Lemma 1, because $\texttt{consistent}(state, xs, ys, \texttt{rev}\, editsRev)$, we can conclude that $\texttt{consistent}(state', xs, ys, \texttt{rev}\, editsRev @ [edit])$, i.e., $\texttt{consistent}(state', xs, ys, \texttt{rev}([edit] @ editsRev)$.

Thus, Lemma 2.1 shows that $\texttt{choices}(state, f, ans)$ returns $ans'@ans$, where ans' is the list of all edit-lists *edits* such that

- $\texttt{prefix}(\texttt{rev}\, editsRev, edits)$,
- $\texttt{annotListToList}(\texttt{doEditsAnnot}(edits, \texttt{listToAnnotList}\, xs)) = ys$,
- $|edits| \leq bound$ and
- $\texttt{noProperPrefixSucceeds}(edits, xs, ys)$,

listed in strictly ascending order, as required.

□

Now we complete the proof of `find`'s correctness. Suppose *editsRev* has type `'a edit list`, *len* is an integer, *state* has type `'a state`, *ans* has type `'a edit list list` and

- $len \leq bound$,
- $len = |editsRev|$,
- $\texttt{noProperPrefixSucceeds}(\texttt{rev}\, editsRev, xs, ys)$ and

- `consistent`$(state, xs, ys,$ `rev` $editsRev)$.

Because $0 \leq len \leq bound$, we have that $bound - len \geq 0$. Thus, applying Lemma 2.2 with $l = bound - len$, we have that `find`$(editsRev, len, state, ans)$ returns $ans'@ans$, where ans' is the list of all edit-lists $edits$ such that

- `prefix`(`rev` $editsRev, edits$),
- `annotListToList`(`doEditsAnnot`$(edits,$ `listToAnnotList` $xs)) = ys$,
- $|edits| \leq bound$ and
- `noProperPrefixSucceeds`$(edits, xs, ys)$,

listed in strictly ascending order, as required.

Now, to complete the proof of `boundedFind`'s correctness, we must show that `find`$([], 0,$ `initialState`$(xs, ys), [])$ returns the list of all edit-lists $edits$ such that

- `annotListToList`(`doEditsAnnot`$(edits,$ `listToAnnotList` $xs)) = ys$,
- $|edits| \leq bound$ and
- `noProperPrefixSucceeds`$(edits, xs, ys)$,

listed in strictly ascending order. Because $bound \geq 0$, we have that $0 \leq bound$. And, $0 = |[]|$. Because `rev` $[] = []$ has no proper prefixes, it follows that `noProperPrefixSucceeds`(`rev` $[], xs, ys)$. And we are given that `consistent`(`initialState`$(xs, ys), xs, ys, [])$, which is the same as `consistent`(`initialState`$(xs, ys), xs, ys,$ `rev` $[])$. Thus, by the specification of `find`, we have that `find`$([], 0,$ `initialState`$(xs, ys), [])$ returns $ans'@[] = ans'$, where ans' is the list of all edit-lists $edits$ such that

- `prefix`(`rev` $[], edits$),
- `annotListToList`(`doEditsAnnot`$(edits,$ `listToAnnotList` $xs)) = ys$,
- $|edits| \leq bound$, and
- `noProperPrefixSucceeds`$(edits, xs, ys)$,

listed in strictly ascending order. But `prefix`(`rev` $[], edits)$, i.e., `prefix`$([], edits)$, always holds, and so we are done.

Exercise 3

Lemma 3.1
For xs and ys of type `'a list`*, there is an $edits$ of type* `'a edit list` *such that $|edits| = |xs| + |ys|$ and* `doEdits`$(edits, xs) = ys$.

Proof. Let $n \in \mathbb{N}$ and $x_1, \ldots, x_n$ be such that $xs = [x_1; \cdots; x_n]$, and let $m \in \mathbb{N}$ and $y_1, \ldots, y_m$ be such that $ys = [y_1; \cdots; y_m]$. Let $edits_1$ consist of n occurrences of `Delete` 0, and $edits_2$ be

$$[\texttt{Append}\ y_1; \cdots; \texttt{Append}\ y_m].$$

It is easy to see that $|edits_1| = n = |xs|$, $\texttt{doEdits}(edits_1, xs) = [\,]$, $|edits_2| = m = |ys|$, $\texttt{doEdits}(edits_2, [\,]) = ys$. Thus if we let $edits$ be $edits_1$ @ $edits_2$, we have that $|edits| = |xs| + |ys|$ and $\texttt{doEdits}(edits, xs) = ys$. □

Suppose xs and ys have type `'a list`. We must show that $\texttt{minimalEdits}(xs, ys)$ returns the list of all minimal edit-lists taking xs to ys, listed in strictly ascending order.

First, we must prove that the auxiliary function `loop` is correct. Let m be the smallest natural number such that there is an $edits$ of type `'a edit list` such that $|edits| = m$ and $\texttt{doEdits}(edits, xs) = ys$. By Lemma 3.1, m is well-defined.

Lemma 3.2
For all $l \in \mathbb{N}$, for all $n \in \mathbb{N}$, if $m - n = l$, then `loop` *n returns the list of all minimal edit-lists taking xs to ys, listed in strictly ascending order.*

Proof. Define a predicate $P(l)$ on natural numbers by: $P(l)$ iff, for all $n \in \mathbb{N}$, if $m - n = l$, then `loop` n returns the list of all minimal edit-lists taking xs to ys, listed in strictly ascending order. We use ordinary induction to prove that, for all $l \in \mathbb{N}$, $P(l)$.

(Basis Step) We must show $P(0)$. Suppose $n \in \mathbb{N}$ and $m - n = 0$. We must show that `loop` n returns the list of all minimal edit-lists taking xs to ys, listed in strictly ascending order. Because $m - n = 0$, we have that $m = n$. By the specification of `boundedFind`, because $n \geq 0$, we have that $\texttt{boundedFind}(n, xs, ys)$ returns the list, $mins$, of all $edits$ of type `'a edit list` such that

- $\texttt{annotListToList}(\texttt{doEditsAnnot}(edits, \texttt{listToAnnotList}\ xs)) = ys$,
- $|edits| \leq n$ and
- $\texttt{noProperPrefixSucceeds}(edits, xs, ys)$,

listed in strictly ascending order.

Suppose, toward a contradiction, that $mins$ is empty. By the definition of m, there is an $edits$ such that $|edits| = m = n$ and $\texttt{doEdits}(edits, xs) = ys$. Also by m's definition, we have that $\texttt{noProperPrefixSucceeds}(edits, xs, ys)$ holds. Thus we must have that $\texttt{doEditsAnnot}(edits, \texttt{listToAnnotList}\ xs)$ raises `Error`. But by the Fundamental Lemma for Annotated Lists, it follows that there is a shorter edit-list than $edits$ taking xs to ys, contradicting the definition of m.

Thus we have that $mins$ is nonempty, so that `loop` n returns $mins$. So we must show that $mins$ is all the minimal edit-lists taking xs to ys, listed in strictly ascending order. We have that the elements of $mins$ are listed in strictly ascending order.

Suppose $edits$ is in $mins$. We must show that $edits$ is a minimal edit-list taking xs to ys. Because $\texttt{annotListToList}(\texttt{doEditsAnnot}(edits, \texttt{listToAnnotList}\ xs)) = ys$, we have that

`doEdits`$(edits, xs) = ys$. We know that $|edits| \leq n = m$ and that there are no edit-lists taking xs to ys that are strictly shorter than m. Thus $|edits| = m$, so that $edits$ is a minimal edit-list taking xs to ys.

Suppose $edits$ is a minimal edit-list taking xs to ys. We must show that $edits$ is in $mins$. By the definition of m, we have that $|edits| = m = n$, so that $|edits| \leq n$. And `noProperPrefixSucceeds`$(edits, xs, ys)$ holds because otherwise there would be a shorter edit-list taking xs to ys. So it remains to show that `annotListToList`(`doEditsAnnot`$(edits,$ `listToAnnotList` $xs)) = ys$. Suppose, toward a contradiction, that this fails. Thus `doEditsAnnot`$(edits,$ `listToAnnotList` $xs)$ raises `Error`. By the Fundamental Lemma for Annotated Lists, it follows that there is a shorter edit-list than $edits$ taking xs to ys—contradicting the definition of m. Thus `annotListToList`(`doEditsAnnot`$(edits,$ `listToAnnotList` $xs)) = ys$.

(Inductive Step) Suppose $l \in \mathbb{N}$, and assume the inductive hypothesis, $P(l)$. We must show that $P(l+1)$. Suppose $n \in \mathbb{N}$ and $m - n = l + 1$. We must show that `loop` n returns the list of all minimal edit-lists taking xs to ys, listed in strictly ascending order. Because $l, m, n \in \mathbb{N}$ and $m - n = l + 1$, it follows that $n < m$. Thus, by the definition of m, there are no edit-lists of length no more than n taking xs to ys. By the specification of `boundedFind`, because $n \in \mathbb{N}$, we have that `boundedFind`(n, xs, ys) returns the list, $mins$, of all $edits$ of type `'a edit list` such that

- `annotListToList`(`doEditsAnnot`$(edits,$ `listToAnnotList` $xs)) = ys$,
- $|edits| \leq n$ and
- `noProperPrefixSucceeds`$(edits, xs, ys)$,

listed in strictly ascending order. Thus $mins$ is empty, so that `loop` n simplifies to `loop`$(n+1)$, and so it will suffice to show that `loop`$(n+1)$ returns the list of all minimal edit-lists taking xs to ys, listed in strictly ascending order. By the inductive hypothesis, $P(l)$, we have that, if $m - (n+1) = l$, then `loop`$(n+1)$ returns the list of all minimal edit-lists taking xs to ys, listed in strictly ascending order. Since $m - n = l + 1$, it follows that $m - (n+1) = l$. Hence `loop`$(n+1)$ returns the list of all minimal edit-lists taking xs to ys, listed in strictly ascending order, as required.

□

To finish the proof of `loop`'s correctness, suppose n is an integer, $n \geq 0$ and there is no $edits$ of type `'a edit list` such that $|edits| < n$ and `doEdits`$(edits, xs) = ys$. Then $n \leq m$, by the definition of m, so that $m - n \geq 0$. Applying Lemma 3.2 with $l = m - n$ and $n = n$, we have that `loop` n returns the list of all minimal edit-lists taking xs to ys, listed in strictly ascending order, as required.

Now we return to showing that `minimalEdits`(xs, ys) returns the list of all minimal edit-lists taking xs to ys, listed in strictly ascending order. It will suffice to show that `loop` 0 returns the list of all minimal edit-lists taking xs to ys, listed in strictly ascending order. Clearly, $0 \geq 0$. And there is no $edits$ of type `'a edit list` of length strictly less than 0. Thus, by the specification of `loop`, we have the desired result.