Proving Properties of Programs

Correctness of Insertion Sort

PLT @ Northwestern

Computer Science, Northwestern University

Back to Insertion Sort

- output is ordered
- output is a permutation of the input

```
(define (sort l)
  (match l
    ['()           l]
    [(cons hd tl) (insert hd (sort tl))]))

(define (insert x l)
  (match l
    ['()           (cons x '())]
    [(cons hd tl) (if (< x hd)
                      (cons x l)
                      (cons hd (insert x tl)))]))
```

An Attempt to Define Sortedness

Property: for all lists l, $\mathit{Sorted}((\texttt{sort}\ l))$

An Attempt to Define Sortedness

Property: for all lists l, $Sorted($`(sort` l`)`$)$

Definition Attempt (Sortedness). A list

`(cons` x_1 `(cons` x_2 `...(cons` x_n `'()))))`

is sorted if $x_1 \leq x_2 \leq \cdots \leq x_{n-1} \leq x_n$.

An Attempt to Define Sortedness

Property: for all lists l, $\mathit{Sorted}((\texttt{sort}\ l))$

Definition Attempt (Sortedness). A list

$$\texttt{(cons}\ x_1\ \texttt{(cons}\ x_2\ \texttt{...(cons}\ x_n\ \texttt{'()))))}$$

is sorted if $x_1 \leq x_2 \leq \cdots \leq x_{n-1} \leq x_n$.

We want a definition that is...

- Intuitively expressing “sortedness”
- Easy to check in math
- Easy to work with, to communicate and to derive other properties

An Attempt to Define Sortedness

Property: for all lists l, $\mathit{Sorted}(\texttt{(sort } l\texttt{)})$

Definition Attempt (Sortedness). A list

$$\texttt{(cons } x_1 \texttt{ (cons } x_2 \texttt{ ...(cons } x_n \texttt{ '()))))}$$

is sorted if $x_1 \leq x_2 \leq \cdots \leq x_{n-1} \leq x_n$.

We want a definition that is...

- Intuitively expressing "sortedness"
- Easy to check in math
- Easy to work with, to communicate and to derive other properties
- **Drawback:** inconvenient — list functions are defined by recursion

Ornamenting the Data Definition of List

Recall: a list l is either:

- An empty list `'()`
- A `cons` cell `(cons` y l'`)` where l' is another list.

Therefore we define $\mathit{Sorted}(l) := \mathit{Increasing}(-\infty, l)$ and $\mathit{Increasing}(L, l)$ as

Ornamenting the Data Definition of List

Recall: a list l is either:

- An empty list `'()`
- A `cons` cell `(cons` y l'`)` where l' is another list.

Therefore we define $Sorted(l) := Increasing(-\infty, l)$ and $Increasing(L, l)$ as

- (For the case $l :=$ `'()`)

  [I-Empty]: $Increasing(L,$ `'()`$)$ is true.

Ornamenting the Data Definition of List

Recall: a list l is either:

- An empty list `'()`
- A `cons` cell `(cons` y l'`)` where l' is another list.

Therefore we define $Sorted(l) := Increasing(-\infty, l)$ and $Increasing(L, l)$ as

- (For the case $l :=$ `'()`)

  [I-EMPTY]: $Increasing(L,$ `'()`$)$ is true.

- (For the case $l :=$ `(cons` y l'`)`)

  [I-CONS]: for all L, l' and y, if $L \leq y$ and $Increasing(y, l')$ then $Increasing(L,$ `(cons` y l'`)`$)$.

Ornamenting the Data Definition of List: Example

- [I-Empty]: $\mathit{Increasing}(L, \texttt{'()})$ is true.
- [I-Cons]: for all L, l' and y, if $L \leq y$ and $\mathit{Increasing}(y, l')$ then $\mathit{Increasing}(L, \texttt{(cons } y\ l' \texttt{)})$.

Example. `(cons 3 (cons 4 (cons 9 '())))` is sorted.

Ornamenting the Data Definition of List: Example

- [I-EMPTY]: $\mathit{Increasing}(L, \texttt{'()})$ is true.
- [I-CONS]: for all L, l' and y, if $L \leq y$ and $\mathit{Increasing}(y, l')$ then $\mathit{Increasing}(L, \texttt{(cons } y\ l'\texttt{)})$.

Example. `(cons 3 (cons 4 (cons 9 '())))` is sorted.

1. $\mathit{Increasing}(9, \texttt{'()})$ by [I-EMPTY]

Ornamenting the Data Definition of List: Example

- [I-EMPTY]: $\mathit{Increasing}(L, \texttt{'()})$ is true.
- [I-CONS]: for all L, l' and y, if $L \leq y$ and $\mathit{Increasing}(y, l')$ then $\mathit{Increasing}(L, \texttt{(cons } y\ l'\texttt{)})$.

Example. `(cons 3 (cons 4 (cons 9 '())))` is sorted.

1. $\mathit{Increasing}(9, \texttt{'()})$ by [I-EMPTY]
2. $\mathit{Increasing}(4, \texttt{(cons 9 '())})$ by [I-CONS]

Ornamenting the Data Definition of List: Example

- [I-Empty]: $\mathit{Increasing}(L,$ `'()`$)$ is true.
- [I-Cons]: for all L, l' and y, if $L \leq y$ and $\mathit{Increasing}(y, l')$ then $\mathit{Increasing}(L,$ `(cons` $y\ l'$`)`$)$.

Example. `(cons 3 (cons 4 (cons 9 '())))` is sorted.

1. $\mathit{Increasing}(9,$ `'()`$)$ by [I-Empty]
2. $\mathit{Increasing}(4,$ `(cons 9 '())`$)$ by [I-Cons]
   because $4 \leq 9$ and $\mathit{Increasing}(9,$ `'()`$)$ by (1)

Ornamenting the Data Definition of List: Example

- [I-EMPTY]: $\mathit{Increasing}(L, \texttt{'()})$ is true.
- [I-CONS]: for all L, l' and y, if $L \leq y$ and $\mathit{Increasing}(y, l')$ then $\mathit{Increasing}(L, \texttt{(cons } y\ l' \texttt{)})$.

Example. `(cons 3 (cons 4 (cons 9 '())))` is sorted.

1. $\mathit{Increasing}(9, \texttt{'()})$ by [I-EMPTY]
2. $\mathit{Increasing}(4, \texttt{(cons 9 '())})$ by [I-CONS]
   because $4 \leq 9$ and $\mathit{Increasing}(9, \texttt{'()})$ by (1)
3. $\mathit{Increasing}(3, \texttt{(cons 4 (cons 9 '()))})$ by [I-CONS]
   because $3 \leq 4$ and $\mathit{Increasing}(4, \texttt{(cons 9 '())})$ by (2)

Ornamenting the Data Definition of List: Example

- [I-EMPTY]: $Increasing(L,$ `'()`$)$ is true.
- [I-CONS]: for all L, l' and y, if $L \leq y$ and $Increasing(y, l')$ then $Increasing(L,$ `(cons` y l'`)`$)$.

Example. `(cons 3 (cons 4 (cons 9 '())))` is sorted.

1. $Increasing(9,$ `'()`$)$ by [I-EMPTY]
2. $Increasing(4,$ `(cons 9 '())`$)$ by [I-CONS]
   because $4 \leq 9$ and $Increasing(9,$ `'()`$)$ by (1)
3. $Increasing(3,$ `(cons 4 (cons 9 '()))`$)$ by [I-CONS]
   because $3 \leq 4$ and $Increasing(4,$ `(cons 9 '())`$)$ by (2)
4. $Increasing(-\infty,$ `(cons 3 (cons 4 (cons 9 '())))`$)$ by [I-CONS]
   because $-\infty \leq 3$ and $Increasing(3,$ `(cons 4 (cons 9 '()))`$)$ by (3)

Ornamenting the Data Definition of List: Inversion

- [I-EMPTY]: $Increasing(L, \texttt{'()})$ is true.
- [I-CONS]: for all L, l' and y, if $L \le y$ and $Increasing(y, l')$ then $Increasing(L, \texttt{(cons } y\ l'\texttt{)})$.

If $Increasing(L, l)$, what can we derive about L and l?

Ornamenting the Data Definition of List: Inversion

- [I-EMPTY]: $\mathit{Increasing}(L, \texttt{'()})$ is true.
- [I-CONS]: for all L, l' and y, if $L \leq y$ and $\mathit{Increasing}(y, l')$ then $\mathit{Increasing}(L, \texttt{(cons } y\ l'\texttt{)})$.

If $\mathit{Increasing}(L, l)$, what can we derive about L and l?

$\mathit{Increasing}$ is generated by [I-EMPTY] and [I-CONS]. If $\mathit{Increasing}(L, l)$, this fact must come from one of the rules.

Ornamenting the Data Definition of List: Inversion

- [I-Empty]: $\mathit{Increasing}(L, \texttt{'()})$ is true.
- [I-Cons]: for all L, l' and y, if $L \leq y$ and $\mathit{Increasing}(y, l')$ then $\mathit{Increasing}(L, \texttt{(cons } y\ l'\texttt{)})$.

If $\mathit{Increasing}(L, l)$, what can we derive about L and l?

Increasing is generated by [I-Empty] and [I-Cons]. If $\mathit{Increasing}(L, l)$, this fact must come from one of the rules.

Example. $\mathit{Increasing}(3, \texttt{(cons } 4 \texttt{ (cons } 9 \texttt{ '())))})$.
By inversion, $3 \leq 4$ and $\mathit{Increasing}(4, \texttt{(cons } 9 \texttt{ '()))})$.

Ornamenting the Data Definition of List: Inversion

- [I-Empty]: $\mathit{Increasing}(L, \texttt{'()})$ is true.
- [I-Cons]: for all L, l' and y, if $L \leq y$ and $\mathit{Increasing}(y, l')$ then $\mathit{Increasing}(L, \texttt{(cons } y\ l'\texttt{)})$.

If $\mathit{Increasing}(L, l)$, what can we derive about L and l?

$\mathit{Increasing}$ is generated by [I-Empty] and [I-Cons]. If $\mathit{Increasing}(L, l)$, this fact must come from one of the rules.

Example. $\mathit{Increasing}(3, \texttt{(cons 4 (cons 9 '()))})$.
By inversion, $3 \leq 4$ and $\mathit{Increasing}(4, \texttt{(cons 9 '())})$.

Example. If $\mathit{Increasing}(L, \texttt{(cons } y\ l'\texttt{)})$, by inversion we know that $L \leq y$ and $\mathit{Increasing}(y, l')$.

Proving Sortedness by Induction

Property: For all lists l, $Increasing(-\infty, (\texttt{sort}\ l))$

Proving Sortedness by Induction

Property: For all lists l, $Increasing(-\infty, (\texttt{sort}\ l))$

Proof Template. Induction on l.

- Case l is `'()`: show that $Increasing(-\infty, (\texttt{sort '()}))$.
- Case l is $(\texttt{cons}\ y\ l')$: assuming $Increasing(-\infty, (\texttt{sort}\ l'))$, show that $Increasing(-\infty, (\texttt{sort}\ (\texttt{cons}\ y\ l')))$.
- By induction, $Increasing(-\infty, (\texttt{sort}\ l))$ holds for all lists l.

Similar to the `length-append` example, we want to prove $Increasing(-\infty, (\texttt{sort}\ (\texttt{cons}\ y\ l')))$.

Proving Sortedness by Induction

Property: For all lists l, $\mathit{Increasing}(-\infty, \texttt{(sort } l\texttt{)})$

Proof Template. Induction on l.

- Case l is `'()`: show that $\mathit{Increasing}(-\infty, \texttt{(sort '())})$.
- Case l is $\texttt{(cons } y\ l'\texttt{)}$: assuming $\mathit{Increasing}(-\infty, \texttt{(sort } l'\texttt{)})$, show that $\mathit{Increasing}(-\infty, \texttt{(sort (cons } y\ l'\texttt{)))})$.
- By induction, $\mathit{Increasing}(-\infty, \texttt{(sort } l\texttt{)})$ holds for all lists l.

Similar to the `length-append` example, we want to prove $\mathit{Increasing}(-\infty, \texttt{(sort (cons } y\ l'\texttt{)))})$.

- [I-Cons]: for all L, l' and y, if $L \leq y$ and $\mathit{Increasing}(y, l')$ then $\mathit{Increasing}(L, \texttt{(cons } y\ l'\texttt{)})$.

Turn $\texttt{(sort (cons } y\ l'\texttt{))}$ into `'()` or `(cons _ _)`!

Running the Sort Function

```
(define (sort l)
  (match l
    ['()            l]
    [(cons hd tl) (insert hd (sort tl))]))
```

Running `sort` on `(cons` y l'`)`:

(sort (cons y l'))

$\equiv$

Running the Sort Function

```
(define (sort l)
  (match l
    ['()            l]
    [(cons hd tl) (insert hd (sort tl))]))
```

Running `sort` on `(cons` y l'`)`:

```
  (sort (cons y l'))
≡                    (the rule of function call)
  (match (cons y l')
    ['()            (cons y l')]
    [(cons hd tl) (insert hd (sort tl))])
≡
```

Running the Sort Function

```
(define (sort l)
  (match l
    ['()            l]
    [(cons hd tl) (insert hd (sort tl))]))
```

Running `sort` on `(cons` y l'`)`:

  `(sort (cons` y l'`))`

$\equiv$ *(the rule of function call)*

  `(match (cons` y l'`)`
    `['()` `(cons` y l'`)]`
    `[(cons hd tl) (insert hd (sort tl))])`

$\equiv$ *(the rules of `match`)*

  `(insert` y `(sort` l'`))`

The Induction Step in the Proof of Sortedness

For any l' and y, assuming $\mathit{Increasing}(-\infty, (\texttt{sort}\ l'))$, we want to show that $\mathit{Increasing}(-\infty, (\texttt{sort}\ (\texttt{cons}\ y\ l')))$.

The Induction Step in the Proof of Sortedness

For any l' and y, assuming $Increasing(-\infty, \texttt{(sort } l'\texttt{)})$, we want to show that $Increasing(-\infty, \texttt{(sort (cons } y\ l'\texttt{))})$.

By calculation,

```
    (sort (cons y l'))
≡
    (match (cons y l')
      ['()           (cons y l')]
      [(cons hd tl) (insert hd (sort tl))])
≡
    (insert y (sort l'))
```

The Induction Step in the Proof of Sortedness

For any l' and y, assuming $\mathit{Increasing}(-\infty, (\texttt{sort}\ l'))$, we want to show that $\mathit{Increasing}(-\infty, (\texttt{sort}\ (\texttt{cons}\ y\ l')))$.

By calculation,

```
    (sort (cons y l'))
≡
    (match (cons y l')
      ['()          (cons y l')]
      [(cons hd tl) (insert hd (sort tl))])
≡
    (insert y (sort l'))
```

Therefore we need to prove $\mathit{Increasing}(-\infty, (\texttt{insert}\ y\ (\texttt{sort}\ l')))$.

Lemma: Insertion Preserves Sortedness

```
(define (insert x l)
  (match l
    ['()           (cons x '())]
    [(cons hd tl) (if (< x hd)
                      (cons x l)
                      (cons hd (insert x tl)))]))
```

Lemma: for any l and x, if $Increasing(-\infty, l)$ then $Increasing(-\infty,$ `(insert` x l`)`$)$.

Lemma: Insertion Preserves Sortedness

```
(define (insert x l)
  (match l
    ['()             (cons x '())]
    [(cons hd tl) (if (< x hd)
                      (cons x l)
                      (cons hd (insert x tl)))]))
```

Lemma: for any l and x, if $Increasing(-\infty, l)$ then $Increasing(-\infty,$ `(insert` x l`)`$)$.

Proof Obligations. Induction on l.

- Case l is `'()`: show that $Increasing(-\infty,$ `(insert` x `'()`$)$`)`.
- Case l is `(cons` y l'`)`: ~~assuming "for any z, if $Increasing(-\infty, l')$ then $Increasing(-\infty,$ `(insert` z l'`)`$)$"~~, show that $Increasing(-\infty,$ `(insert` x `(cons` y l'`))`$)$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $Increasing(L, l)$ then $Increasing(L, (\texttt{insert}\ x\ l))$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $\mathit{Increasing}(L, l)$ then $\mathit{Increasing}(L, (\texttt{insert}\ x\ l))$.

Proof Obligations. Induction on l. Assume that $L \leq x$ and $\mathit{Increasing}(L, l)$.

- Case l is `'()`: show that $\mathit{Increasing}(L, (\texttt{insert}\ x\ \texttt{'()}))$.

- Case l is $(\texttt{cons}\ y\ l')$: assuming
  "for any M and z, if $M \leq z$ and $\mathit{Increasing}(M, l')$
  then $\mathit{Increasing}(M, (\texttt{insert}\ z\ l'))$,"
  show that $\mathit{Increasing}(L, (\texttt{insert}\ x\ (\texttt{cons}\ y\ l')))$.

- By induction, $\mathit{Increasing}(L, (\texttt{insert}\ x\ l))$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $\mathit{Increasing}(L, l)$ then $\mathit{Increasing}(L, (\texttt{insert}\ x\ l))$.

Proof (1/3). Induction on l. Assume that $L \leq x$ and $\mathit{Increasing}(L, l)$.

- Case l is `'()`: show that $\mathit{Increasing}(L, (\texttt{insert}\ x\ \texttt{'()}))$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $Increasing(L, l)$ then $Increasing(L, (\texttt{insert}\ x\ l))$.

Proof (1/3). Induction on l. Assume that $L \leq x$ and $Increasing(L, l)$.

- Case l is `'()`: show that $Increasing(L, (\texttt{insert}\ x\ \texttt{'()}))$.
  - By calculation, $(\texttt{insert}\ x\ \texttt{'()}) \equiv (\texttt{cons}\ x\ \texttt{'()})$. Therefore our proof obligation simplifies to $Increasing(L, (\texttt{cons}\ x\ \texttt{'()}))$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \le x$ and $\mathit{Increasing}(L, l)$ then $\mathit{Increasing}(L, (\texttt{insert}\ x\ l))$.

Proof (1/3). Induction on l. Assume that $L \le x$ and $\mathit{Increasing}(L, l)$.

- Case l is `'()`: show that $\mathit{Increasing}(L, (\texttt{insert}\ x\ \texttt{'()}))$.
  - By calculation, $(\texttt{insert}\ x\ \texttt{'()}) \equiv (\texttt{cons}\ x\ \texttt{'()})$. Therefore our proof obligation simplifies to $\mathit{Increasing}(L, (\texttt{cons}\ x\ \texttt{'()}))$.
  - By [I-Cons], $\mathit{Increasing}(L, (\texttt{cons}\ x\ \texttt{'()}))$ if $L \le x$ and $\mathit{Increasing}(x, \texttt{'()})$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $Increasing(L, l)$ then $Increasing(L, (\texttt{insert}\ x\ l))$.

Proof (1/3). Induction on l. Assume that $L \leq x$ and $Increasing(L, l)$.

- Case l is `'()`: show that $Increasing(L, (\texttt{insert}\ x\ \texttt{'()}))$.
  - By calculation, $(\texttt{insert}\ x\ \texttt{'()}) \equiv (\texttt{cons}\ x\ \texttt{'()})$. Therefore our proof obligation simplifies to $Increasing(L, (\texttt{cons}\ x\ \texttt{'()}))$.
  - By [I-Cons], $Increasing(L, (\texttt{cons}\ x\ \texttt{'()}))$ if $L \leq x$ and $Increasing(x, \texttt{'()})$.
    - By assumption, $L \leq x$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $Increasing(L, l)$ then $Increasing(L, (\texttt{insert}\ x\ l))$.

Proof (1/3). Induction on l. Assume that $L \leq x$ and $Increasing(L, l)$.

- Case l is `'()`: show that $Increasing(L, (\texttt{insert}\ x\ \texttt{'()}))$.
  - By calculation, $(\texttt{insert}\ x\ \texttt{'()}) \equiv (\texttt{cons}\ x\ \texttt{'()})$. Therefore our proof obligation simplifies to $Increasing(L, (\texttt{cons}\ x\ \texttt{'()}))$.
  - By [I-CONS], $Increasing(L, (\texttt{cons}\ x\ \texttt{'()}))$ if $L \leq x$ and $Increasing(x, \texttt{'()})$.
    - By assumption, $L \leq x$.
    - By [I-EMPTY], $Increasing(x, \texttt{'()})$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $Increasing(L, l)$ then $Increasing(L, (\texttt{insert}\ x\ l))$.

Proof (2/3). Induction on l. Assume that $L \leq x$ and $Increasing(L, l)$.

- Case l is $(\texttt{cons}\ y\ l')$: assuming

  "for any M and z, if $M \leq z$ and $Increasing(M, l')$
  then $Increasing(M, (\texttt{insert}\ z\ l'))$,"

  show that $Increasing(L, (\texttt{insert}\ x\ (\texttt{cons}\ y\ l')))$.

Running the Insertion Function

To show $\mathit{Increasing}(L,$ `(insert` x `(cons` y l'`))`$)$, we calculate the result of running `(insert` x `(cons` y l'`))`:

Running the Insertion Function

To show $\mathit{Increasing}(L,$ `(insert` x `(cons` y l'`))`$)$, we calculate the result of running `(insert` x `(cons` y l'`))`:

```
  (insert x (cons y l'))
≡               (the rule of function call)
  (match (cons y l')
    ['()            (cons x '())]
    [(cons hd tl) (if (< x hd)
                      (cons x (cons y l'))
                      (cons hd (insert x tl)))])
≡               (the rules of match)
  (if  (< x y)  (cons x (cons y l'))  (cons y (insert x l')))
```

Running the Insertion Function

To show $Increasing(L,$ `(insert` x `(cons` y l'`))`$)$, we calculate the result of running `(insert` x `(cons` y l'`))`:

```
  (insert x (cons y l'))
≡              (the rule of function call)
  (match (cons y l')
    ['()             (cons x '())]
    [(cons hd tl) (if (< x hd)
                      (cons x (cons y l'))
                      (cons hd (insert x tl)))])
≡              (the rules of match)
  (if (< x y) (cons x (cons y l')) (cons y (insert x l')))
```

Our current assumptions: $L \leq x$ and $Increasing(L, l)$

Running the Insertion Function

Take cases on the comparison result. If $x < y$,

$$
\begin{aligned}
& \texttt{(insert } x \texttt{ (cons } y\ l' \texttt{))} \\
\equiv & \qquad\qquad \textit{(the previous page)} \\
& \texttt{(if (< } x\ y \texttt{) (cons } x \texttt{ (cons } y\ l' \texttt{)) (cons } y \texttt{ (insert } x\ l' \texttt{)))} \\
\equiv &
\end{aligned}
$$

Running the Insertion Function

Take cases on the comparison result. If $x < y$,

$$
\begin{aligned}
& \texttt{(insert } x \texttt{ (cons } y\ l' \texttt{))} \\
\equiv & \quad \textit{(the previous page)} \\
& \texttt{(if (< } x\ y \texttt{) (cons } x \texttt{ (cons } y\ l' \texttt{)) (cons } y \texttt{ (insert } x\ l' \texttt{)))} \\
\equiv & \quad \textit{(arithmetic and the additional assumption that } x < y \textit{)} \\
& \texttt{(if \#t (cons } x \texttt{ (cons } y\ l' \texttt{)) (cons } y \texttt{ (insert } x\ l' \texttt{)))}
\end{aligned}
$$

Running the Insertion Function

Take cases on the comparison result. If $x < y$,

$$
\begin{aligned}
& \texttt{(insert } x \texttt{ (cons } y\ l'\texttt{))} \\
\equiv \quad & \textit{(the previous page)} \\
& \texttt{(if (< } x\ y\texttt{) (cons } x \texttt{ (cons } y\ l'\texttt{)) (cons } y \texttt{ (insert } x\ l'\texttt{)))} \\
\equiv \quad & \textit{(arithmetic and the additional assumption that } x < y\textit{)} \\
& \texttt{(if \#t (cons } x \texttt{ (cons } y\ l'\texttt{)) (cons } y \texttt{ (insert } x\ l'\texttt{)))} \\
\equiv \quad & \textit{(the rules of } \texttt{if}\textit{)} \\
& \texttt{(cons } x \texttt{ (cons } y\ l'\texttt{))}
\end{aligned}
$$

Rules of `if`:*

$$\texttt{(if \#t } e_1\ e_2\texttt{)} \equiv e_1 \qquad\qquad \texttt{(if \#f } e_1\ e_2\texttt{)} \equiv e_2$$

*In Racket, any non-`#f` value has the same effect as `#t` in `if` expressions. Here we make the simplifying assumption that `if` expressions must take only boolean values.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $Increasing(L, l)$ then $Increasing(L, (\texttt{insert}\ x\ l))$.

Proof (2/3). Induction on l. Assume that $L \leq x$ and $Increasing(L, l)$.

- Case l is $(\texttt{cons}\ y\ l')$: assuming

  "for any M and z, if $M \leq z$ and $Increasing(M, l')$
  then $Increasing(M, (\texttt{insert}\ z\ l'))$,"

  show that $Increasing(L, (\texttt{insert}\ x\ (\texttt{cons}\ y\ l')))$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $Increasing(L, l)$ then $Increasing(L, (\texttt{insert}\ x\ l))$.

Proof (2/3). Induction on l. Assume that $L \leq x$ and $Increasing(L, l)$.

- Case l is $(\texttt{cons}\ y\ l')$: assuming

  "for any M and z, if $M \leq z$ and $Increasing(M, l')$
  then $Increasing(M, (\texttt{insert}\ z\ l'))$,"

  show that $Increasing(L, (\texttt{insert}\ x\ (\texttt{cons}\ y\ l')))$.

  - Case $x < y$: by calculation, the proof obligation simplifies to $Increasing(L, (\texttt{cons}\ x\ (\texttt{cons}\ y\ l')))$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $\mathit{Increasing}(L, l)$ then $\mathit{Increasing}(L, (\texttt{insert}\ x\ l))$.

Proof (2/3). Induction on l. Assume that $L \leq x$ and $\mathit{Increasing}(L, l)$.

- Case l is $(\texttt{cons}\ y\ l')$: assuming

  "for any M and z, if $M \leq z$ and $\mathit{Increasing}(M, l')$
  then $\mathit{Increasing}(M, (\texttt{insert}\ z\ l'))$,"

  show that $\mathit{Increasing}(L, (\texttt{insert}\ x\ (\texttt{cons}\ y\ l')))$.
  - Case $x < y$: by calculation, the proof obligation simplifies to $\mathit{Increasing}(L, (\texttt{cons}\ x\ (\texttt{cons}\ y\ l')))$.
  - By assumption, $\mathit{Increasing}(L, l)$, i.e. $\mathit{Increasing}(L, (\texttt{cons}\ y\ l'))$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $\mathit{Increasing}(L, l)$ then $\mathit{Increasing}(L, (\texttt{insert}\ x\ l))$.

Proof (2/3). Induction on l. Assume that $L \leq x$ and $\mathit{Increasing}(L, l)$.

- Case l is $(\texttt{cons}\ y\ l')$: assuming

  "for any M and z, if $M \leq z$ and $\mathit{Increasing}(M, l')$
  then $\mathit{Increasing}(M, (\texttt{insert}\ z\ l'))$,"

  show that $\mathit{Increasing}(L, (\texttt{insert}\ x\ (\texttt{cons}\ y\ l')))$.
  - Case $x < y$: by calculation, the proof obligation simplifies to $\mathit{Increasing}(L, (\texttt{cons}\ x\ (\texttt{cons}\ y\ l')))$.
  - By assumption, $\mathit{Increasing}(L, l)$, i.e. $\mathit{Increasing}(L, (\texttt{cons}\ y\ l'))$.
  - By inversion, $L \leq y$ and $\mathit{Increasing}(y, l')$ due to [I-Cons].

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $\mathit{Increasing}(L, l)$ then $\mathit{Increasing}(L, (\texttt{insert}\ x\ l))$.

Proof (2/3). Induction on l. Assume that $L \leq x$ and $\mathit{Increasing}(L, l)$.

- Case l is $(\texttt{cons}\ y\ l')$: assuming
  "for any M and z, if $M \leq z$ and $\mathit{Increasing}(M, l')$
  then $\mathit{Increasing}(M, (\texttt{insert}\ z\ l'))$,"

  show that $\mathit{Increasing}(L, (\texttt{insert}\ x\ (\texttt{cons}\ y\ l')))$.
  - Case $x < y$: by calculation, the proof obligation simplifies to $\mathit{Increasing}(L, (\texttt{cons}\ x\ (\texttt{cons}\ y\ l')))$.
  - By assumption, $\mathit{Increasing}(L, l)$, i.e. $\mathit{Increasing}(L, (\texttt{cons}\ y\ l'))$.
  - By inversion, $L \leq y$ and $\mathit{Increasing}(y, l')$ due to [I-Cons].
  - With $x < y$ we obtain $\mathit{Increasing}(x, (\texttt{cons}\ y\ l'))$ through [I-Cons].

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $\mathit{Increasing}(L, l)$ then $\mathit{Increasing}(L, (\texttt{insert}\ x\ l))$.

Proof (2/3). Induction on l. Assume that $L \leq x$ and $\mathit{Increasing}(L, l)$.

- Case l is $(\texttt{cons}\ y\ l')$: assuming

  "for any M and z, if $M \leq z$ and $\mathit{Increasing}(M, l')$
  then $\mathit{Increasing}(M, (\texttt{insert}\ z\ l'))$,"

  show that $\mathit{Increasing}(L, (\texttt{insert}\ x\ (\texttt{cons}\ y\ l')))$.
  - Case $x < y$: by calculation, the proof obligation simplifies to $\mathit{Increasing}(L, (\texttt{cons}\ x\ (\texttt{cons}\ y\ l')))$.
  - By assumption, $\mathit{Increasing}(L, l)$, i.e. $\mathit{Increasing}(L, (\texttt{cons}\ y\ l'))$.
  - By inversion, $L \leq y$ and $\mathit{Increasing}(y, l')$ due to [I-Cons].
  - With $x < y$ we obtain $\mathit{Increasing}(x, (\texttt{cons}\ y\ l'))$ through [I-Cons].
  - By assumption, $L \leq x$. Therefore $\mathit{Increasing}(L, (\texttt{cons}\ x\ (\texttt{cons}\ y\ l')))$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $Increasing(L, l)$ then $Increasing(L, (\texttt{insert}\ x\ l))$.

Proof (3/3). Induction on l. Assume that $L \leq x$ and $Increasing(L, l)$.

- Case l is $(\texttt{cons}\ y\ l')$: assuming

  "for any M and z, if $M \leq z$ and $Increasing(M, l')$
  then $Increasing(M, (\texttt{insert}\ z\ l'))$,"

  show that $Increasing(L, (\texttt{insert}\ x\ (\texttt{cons}\ y\ l')))$.

  - Case $x \geq y$: by calculation, the proof obligation simplifies to $Increasing(L, (\texttt{cons}\ y\ (\texttt{insert}\ x\ l')))$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $Increasing(L, l)$ then $Increasing(L, (\texttt{insert}\ x\ l))$.

Proof (3/3). Induction on l. Assume that $L \leq x$ and $Increasing(L, l)$.

- Case l is $(\texttt{cons}\ y\ l')$: assuming

  "for any M and z, if $M \leq z$ and $Increasing(M, l')$
  then $Increasing(M, (\texttt{insert}\ z\ l'))$,"

  show that $Increasing(L, (\texttt{insert}\ x\ (\texttt{cons}\ y\ l')))$.

  - Case $x \geq y$: by calculation, the proof obligation simplifies to $Increasing(L, (\texttt{cons}\ y\ (\texttt{insert}\ x\ l')))$.
  - By inverting $Increasing(L, l)$, we know that $L \leq y$ and $Increasing(y, l')$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $Increasing(L, l)$ then $Increasing(L, (\texttt{insert}\ x\ l))$.

Proof (3/3). Induction on l. Assume that $L \leq x$ and $Increasing(L, l)$.

- Case l is $(\texttt{cons}\ y\ l')$: assuming

  "for any M and z, if $M \leq z$ and $Increasing(M, l')$
  then $Increasing(M, (\texttt{insert}\ z\ l'))$,"

  show that $Increasing(L, (\texttt{insert}\ x\ (\texttt{cons}\ y\ l')))$.
  - Case $x \geq y$: by calculation, the proof obligation simplifies to $Increasing(L, (\texttt{cons}\ y\ (\texttt{insert}\ x\ l')))$.
  - By inverting $Increasing(L, l)$, we know that $L \leq y$ and $Increasing(y, l')$.
  - Together with $y \leq x$, by induction, $Increasing(y, (\texttt{insert}\ x\ l'))$.

Lemma: Insertion Preserves Lower Bound

Lemma: for any l, L and x, if $L \leq x$ and $\mathit{Increasing}(L, l)$ then $\mathit{Increasing}(L, (\texttt{insert}\ x\ l))$.

Proof (3/3). Induction on l. Assume that $L \leq x$ and $\mathit{Increasing}(L, l)$.

- Case l is $(\texttt{cons}\ y\ l')$: assuming

  "for any M and z, if $M \leq z$ and $\mathit{Increasing}(M, l')$
  then $\mathit{Increasing}(M, (\texttt{insert}\ z\ l'))$,"

  show that $\mathit{Increasing}(L, (\texttt{insert}\ x\ (\texttt{cons}\ y\ l')))$.
  - Case $x \geq y$: by calculation, the proof obligation simplifies to $\mathit{Increasing}(L, (\texttt{cons}\ y\ (\texttt{insert}\ x\ l')))$.
  - By inverting $\mathit{Increasing}(L, l)$, we know that $L \leq y$ and $\mathit{Increasing}(y, l')$.
  - Together with $y \leq x$, by induction, $\mathit{Increasing}(y, (\texttt{insert}\ x\ l'))$.
  - Since $L \leq y$, we have $\mathit{Increasing}(L, (\texttt{cons}\ y\ (\texttt{insert}\ x\ l')))$ by [I-Cons].

Summary: Proving Programs Correct

When proving properties and the correctness of programs…

- We need a model of the programming language
  - Must ensure that the model itself is right. We have been hand-wavy about the correct definition of "≡".
- Phrase the desired properties in terms of the programs in the model
- Induction on the data types used in the programs
- Theorem provers mechanize & automate different aspects involved in the proofs

Summary: Follow Your Data Definition

A list l is one of:

- An empty list `'()`
- A `cons` cell `(cons` y l'`)` where l' is another list.

```
(define (list-function l)
  (match l
    ['() ... the empty case ...]
    [(cons hd tl) ... combine hd and (list-function tl) ...]))
```

Summary: Follow Your Data Definition

$Increasing$ is defined by:

- [I-Empty]: $Increasing(L, \texttt{'()})$ is true.
- [I-Cons]: for all L, l' and y, if $L \leq y$ and $Increasing(y, l')$ then $Increasing(L, \texttt{(cons}\ y\ l'\texttt{)})$.

Proposition. For all lists l, statement of l

Proof (template). Induction on l.

- Case l is `'()`: statement of `'()`
- Case l is $\texttt{(cons}\ y\ l'\texttt{)}$: if statement of l' then statement of $\texttt{(cons}\ y\ l'\texttt{)}$.

By induction, statement of l.

Mechanizing the Correctness Proof

```
insert : ℕ → List ℕ → List ℕ
insert x []          = (x ∷ [])
insert x (hd ∷ tl) with x <? hd
... | true  because _ = x ∷ (hd ∷ tl)
... | false because _ = hd ∷ insert x tl

data Increasing : -∞ℕ → List ℕ → Set where
  [I-Empty] : ∀ {L} → Increasing L []
  [I-Cons]  : ∀ {L y l′} →
    L ≤* [y]ₙ → Increasing [y]ₙ l′ → Increasing L (y ∷ l′)

ins-presrv-lb : ∀ x l {L} →
  L ≤* [x]ₙ → Increasing L l → Increasing L (insert x l)
ins-presrv-lb x []        L≤x [I-Empty] = [I-Cons] L≤x [I-Empty]
ins-presrv-lb x (y ∷ l′)  L≤x ([I-Cons] L≤y l′inc) with x <? y
... | yes x<y = ([I-Cons] L≤x ([I-Cons] [<⇒≤ x<y]ₗ l′inc))
... | no  x≮y = ([I-Cons] L≤y (ins-presrv-lb x l′ [≮⇒≥ x≮y]ₗ l′inc))
```

Proving Properties: Output is a Permutation of the Input

Similar to sortedness, we want to express “is a permutation of” in a workable form. Let $l \leftrightsquigarrow l'$ means that l is a permutation of l'.

Theorem. A list l is a permutation of another list l' if and only if we can obtain l' by repeatedly swapping pairs of elements of l.

Proving Properties: Output is a Permutation of the Input

Similar to sortedness, we want to express "is a permutation of" in a workable form. Let $l \leftrightsquigarrow l'$ means that l is a permutation of l'.

Theorem. A list l is a permutation of another list l' if and only if we can obtain l' by repeatedly swapping pairs of elements of l.

$$
\begin{aligned}
& \underline{4}, 1, 3, \underline{2} \\
\leftrightsquigarrow \quad & \underline{2}, \underline{1}, 3, 4 \\
\leftrightsquigarrow \quad & 1, 2, 3, 4
\end{aligned}
$$

Even better, it is sufficient to consider adjacent pairs of elements.

Permutation as the Product of Adjacent Transpositions

From the theorem, we *define* $l \leftrightsquigarrow l'$ by describing how to repeatedly swap pairs of elements.

- [P-SWAP]: for all x, y and l, $(\texttt{cons}\ \underline{x}\ (\texttt{cons}\ \underline{y}\ l)) \leftrightsquigarrow (\texttt{cons}\ \underline{y}\ (\texttt{cons}\ \underline{x}\ l))$.

Permutation as the Product of Adjacent Transpositions

From the theorem, we *define* $l \leftrightsquigarrow l'$ by describing how to repeatedly swap pairs of elements.

- [P-SWAP]: for all x, y and l, $(\texttt{cons}\ \underline{x}\ (\texttt{cons}\ \underline{y}\ l)) \leftrightsquigarrow (\texttt{cons}\ \underline{y}\ (\texttt{cons}\ \underline{x}\ l))$.
- [P-PREP]: for all y and lists l, l', if $l \leftrightsquigarrow l'$ then $(\texttt{cons}\ y\ l) \leftrightsquigarrow (\texttt{cons}\ y\ l')$

Permutation as the Product of Adjacent Transpositions

From the theorem, we *define* $l \leftrightsquigarrow l'$ by describing how to repeatedly swap pairs of elements.

- [P-SWAP]: for all x, y and l, `(cons` $\underline{x}$ `(cons` $\underline{y}$ l`))` $\leftrightsquigarrow$ `(cons` $\underline{y}$ `(cons` $\underline{x}$ l`))`.
- [P-PREP]: for all y and lists l, l', if $l \leftrightsquigarrow l'$ then `(cons` y l`)` $\leftrightsquigarrow$ `(cons` y l'`)`
- [P-REFL]: for all lists l, $l \leftrightsquigarrow l$.

Permutation as the Product of Adjacent Transpositions

From the theorem, we *define* $l \leftrightsquigarrow l'$ by describing how to repeatedly swap pairs of elements.

- [P-Swap]: for all x, y and l, $(\texttt{cons}\ \underline{x}\ (\texttt{cons}\ \underline{y}\ l)) \leftrightsquigarrow (\texttt{cons}\ \underline{y}\ (\texttt{cons}\ \underline{x}\ l))$.
- [P-Prep]: for all y and lists l, l', if $l \leftrightsquigarrow l'$ then $(\texttt{cons}\ y\ l) \leftrightsquigarrow (\texttt{cons}\ y\ l')$
- [P-Refl]: for all lists l, $l \leftrightsquigarrow l$.
- [P-Trans]: for all lists l, l'' and l', if $l \leftrightsquigarrow l''$ and $l'' \leftrightsquigarrow l'$ then $l \leftrightsquigarrow l'$.

Permutation as the Product of Adjacent Transpositions

- [P-SWAP]: for all x, y and l, `(cons` x `(cons` y l`))` $\leftrightsquigarrow$ `(cons` y `(cons` x l`))`.
- [P-PREP]: for all y and lists l, l', if $l \leftrightsquigarrow l'$ then `(cons` y l`)` $\leftrightsquigarrow$ `(cons` y l'`)`
- [P-REFL]: for all lists l, $l \leftrightsquigarrow l$.
- [P-TRANS]: for all lists l, l'' and l', if $l \leftrightsquigarrow l''$ and $l'' \leftrightsquigarrow l'$ then $l \leftrightsquigarrow l'$.

Ex. `(cons` $\underline{3}$ `(cons` 1 `(cons` 2 `'()))))` $\leftrightsquigarrow$ `(cons` 1 `(cons` 2 `(cons` $\underline{3}$ `'()))))`.

Permutation as the Product of Adjacent Transpositions

- [P-Swap]: for all x, y and l, $\texttt{(cons}\ x\ \texttt{(cons}\ y\ l\texttt{))} \leftrightsquigarrow \texttt{(cons}\ y\ \texttt{(cons}\ x\ l\texttt{))}$.
- [P-Prep]: for all y and lists l, l', if $l \leftrightsquigarrow l'$ then $\texttt{(cons}\ y\ l\texttt{)} \leftrightsquigarrow \texttt{(cons}\ y\ l'\texttt{)}$
- [P-Refl]: for all lists l, $l \leftrightsquigarrow l$.
- [P-Trans]: for all lists l, l'' and l', if $l \leftrightsquigarrow l''$ and $l'' \leftrightsquigarrow l'$ then $l \leftrightsquigarrow l'$.

Ex. `(cons 3 (cons 1 (cons 2 '())))` $\leftrightsquigarrow$ `(cons 1 (cons 2 (cons 3 '())))`.

1. `(cons 3 (cons 1 (cons 2 '())))` $\leftrightsquigarrow$ `(cons 1 (cons 3 (cons 2 '())))`
   by [P-Swap]

Permutation as the Product of Adjacent Transpositions

- [P-SWAP]: for all x, y and l, `(cons` x `(cons` y l`))` $\leftrightsquigarrow$ `(cons` y `(cons` x l`))`.
- [P-PREP]: for all y and lists l, l', if $l \leftrightsquigarrow l'$ then `(cons` y l`)` $\leftrightsquigarrow$ `(cons` y l'`)`
- [P-REFL]: for all lists l, $l \leftrightsquigarrow l$.
- [P-TRANS]: for all lists l, l'' and l', if $l \leftrightsquigarrow l''$ and $l'' \leftrightsquigarrow l'$ then $l \leftrightsquigarrow l'$.

Ex. `(cons 3 (cons 1 (cons 2 '())))` $\leftrightsquigarrow$ `(cons 1 (cons 2 (cons 3 '())))`.

1. `(cons 3 (cons 1 (cons 2 '())))` $\leftrightsquigarrow$ `(cons 1 (cons 3 (cons 2 '())))` by [P-SWAP]
2. `(cons 3 (cons 2 '()))` $\leftrightsquigarrow$ `(cons 2 (cons 3 '()))` by [P-SWAP].

Permutation as the Product of Adjacent Transpositions

- [P-SWAP]: for all x, y and l, `(cons` x `(cons` y l`))` $\leftrightsquigarrow$ `(cons` y `(cons` x l`))`.
- [P-PREP]: for all y and lists l, l', if $l \leftrightsquigarrow l'$ then `(cons` y l`)` $\leftrightsquigarrow$ `(cons` y l'`)`
- [P-REFL]: for all lists l, $l \leftrightsquigarrow l$.
- [P-TRANS]: for all lists l, l'' and l', if $l \leftrightsquigarrow l''$ and $l'' \leftrightsquigarrow l'$ then $l \leftrightsquigarrow l'$.

Ex. `(cons 3 (cons 1 (cons 2 '())))` $\leftrightsquigarrow$ `(cons 1 (cons 2 (cons 3 '())))`.

1. `(cons 3 (cons 1 (cons 2 '())))` $\leftrightsquigarrow$ `(cons 1 (cons 3 (cons 2 '())))` by [P-SWAP]
2. `(cons 3 (cons 2 '()))` $\leftrightsquigarrow$ `(cons 2 (cons 3 '()))` by [P-SWAP].
3. `(cons 1 (cons 3 (cons 2 '())))` $\leftrightsquigarrow$ `(cons 1 (cons 2 (cons 3 '())))` by [P-PREP] and (2).

Permutation as the Product of Adjacent Transpositions

- [P-SWAP]: for all x, y and l, `(cons x (cons y l))` $\leftrightsquigarrow$ `(cons y (cons x l))`.
- [P-PREP]: for all y and lists l, l', if $l \leftrightsquigarrow l'$ then `(cons y l)` $\leftrightsquigarrow$ `(cons y l')`
- [P-REFL]: for all lists l, $l \leftrightsquigarrow l$.
- [P-TRANS]: for all lists l, l'' and l', if $l \leftrightsquigarrow l''$ and $l'' \leftrightsquigarrow l'$ then $l \leftrightsquigarrow l'$.

Ex. `(cons 3 (cons 1 (cons 2 '())))` $\leftrightsquigarrow$ `(cons 1 (cons 2 (cons 3 '())))`.

1. `(cons 3 (cons 1 (cons 2 '())))` $\leftrightsquigarrow$ `(cons 1 (cons 3 (cons 2 '())))` by [P-SWAP]
2. `(cons 3 (cons 2 '()))` $\leftrightsquigarrow$ `(cons 2 (cons 3 '()))` by [P-SWAP].
3. `(cons 1 (cons 3 (cons 2 '())))` $\leftrightsquigarrow$ `(cons 1 (cons 2 (cons 3 '())))` by [P-PREP] and (2).
4. `(cons 3 (cons 1 (cons 2 '())))` $\leftrightsquigarrow$ `(cons 1 (cons 2 (cons 3 '())))` by [P-TRANS], (1) and (3).

Sort Produces a Permutation of its Input

Theorem. For all l, $l \leftrightsquigarrow$ `(sort` l`)`.

Sort Produces a Permutation of its Input

Theorem. For all l, $l \leftrightsquigarrow$ `(sort` l`)`.

Proof Sketch. Induction on l.

- Case l is `'()`: `(sort '())` $\equiv$ `'()`. Therefore, `'()` $\leftrightsquigarrow$ `(sort '())` by [P-Refl].

Sort Produces a Permutation of its Input

Theorem. For all l, $l \leftrightsquigarrow$ `(sort` l`)`.

Proof Sketch. Induction on l.

- Case l is `'()`: `(sort '())` $\equiv$ `'()`. Therefore, `'()` $\leftrightsquigarrow$ `(sort '())` by [P-REFL].

- Case l is `(cons` y l'`)`: assume that $l' \leftrightsquigarrow$ `(sort` l'`)`. Since `(sort (cons` y l'`))` $\equiv$ `(insert` y `(sort` l'`))`, we need to prove

  `(cons` y l'`)` $\leftrightsquigarrow$ `(insert` y `(sort` l'`))`.

By induction, $l \leftrightsquigarrow$ `(sort` l`)` for all l.

Insertion Produces a Permutation of its Input

Lemma. For all l and x, `(cons` x l`)` $\leftrightsquigarrow$ `(insert` x l`)`.

Insertion Produces a Permutation of its Input

Lemma. For all l and x, `(cons` x l`)` $\leftrightsquigarrow$ `(insert` x l`)`.

Proof Sketch. Induction on l.

- Case l is `'()`: we need to show that `(cons` x `'())` $\leftrightsquigarrow$ `(insert` x `'())`.
- Case l is `(cons` y l'`)`: Assume that for all z, `(cons` z l'`)` $\leftrightsquigarrow$ `(insert` z l'`)`. We need to prove that `(cons` x `(cons` y l'`))` $\leftrightsquigarrow$ `(insert` x `(cons` y l'`))`.

Insertion Produces a Permutation of its Input

Lemma. For all l and x, `(cons` x l`)` $\leftrightsquigarrow$ `(insert` x l`)`.

Proof Sketch. Induction on l.

- Case l is `'()`: we need to show that `(cons` x `'())` $\leftrightsquigarrow$ `(insert` x `'())`.
- Case l is `(cons` y l'`)`: Assume that for all z, `(cons` z l'`)` $\leftrightsquigarrow$ `(insert` z l'`)`.
  We need to prove that `(cons` x `(cons` y l'`))` $\leftrightsquigarrow$ `(insert` x `(cons` y l'`))`.
  - Case $x < y$: `(insert` x `(cons` y l'`))` $\equiv$ `(cons` x `(cons` y l'`))`.

Insertion Produces a Permutation of its Input

Lemma. For all l and x, `(cons` x l`)` $\leftrightsquigarrow$ `(insert` x l`)`.

Proof Sketch. Induction on l.

- Case l is `'()`: we need to show that `(cons` x `'())` $\leftrightsquigarrow$ `(insert` x `'())`.
- Case l is `(cons` y l'`)`: Assume that for all z, `(cons` z l'`)` $\leftrightsquigarrow$ `(insert` z l'`)`.
  We need to prove that `(cons` x `(cons` y l'`))` $\leftrightsquigarrow$ `(insert` x `(cons` y l'`))`.
  - Case $x < y$: `(insert` x `(cons` y l'`))` $\equiv$ `(cons` x `(cons` y l'`))`.
  - Case $x \geq y$: `(insert` x `(cons` y l'`))` $\equiv$ `(cons` y `(insert` x l'`))`.

Insertion Produces a Permutation of its Input

Lemma. For all l and x, `(cons` x l`)` $\leftrightsquigarrow$ `(insert` x l`)`.

Proof Sketch. Induction on l.

- Case l is `'()`: we need to show that `(cons` x `'())` $\leftrightsquigarrow$ `(insert` x `'())`.
- Case l is `(cons` y l'`)`: Assume that for all z, `(cons` z l'`)` $\leftrightsquigarrow$ `(insert` z l'`)`. We need to prove that `(cons` x `(cons` y l'`))` $\leftrightsquigarrow$ `(insert` x `(cons` y l'`))`.
  - Case $x < y$: `(insert` x `(cons` y l'`))` $\equiv$ `(cons` x `(cons` y l'`))`.
  - Case $x \geq y$: `(insert` x `(cons` y l'`))` $\equiv$ `(cons` y `(insert` x l'`))`.

	`(cons` $\underline{x}$ `(cons` $\underline{y}$ l'`))`	
$\leftrightsquigarrow$	`(cons` $\underline{y}$ `(cons` $\underline{x}$ l'`))`	By [P-Swap]
$\leftrightsquigarrow$	`(cons` y `(insert` x l'`))`	By [P-Prep] & induction hypothesis