Proving Properties of Programs

What is a Correctness Proof?

PLT @ Northwestern

Computer Science, Northwestern University

Testing Sorting Algorithms

- **output is ordered**

  For all lists l, `(sorted? (sort` l`))`

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

  For all lists l, `(permutation-of?` l `(sort` l`))`

- .. for some sorting algorithms: sort is stable

Proving Programs Correct

- How to state properties of programs?

Proving Programs Correct

- How to state properties of programs?
  - output is ordered: for all lists l, $Sorted(\texttt{(sort } l\texttt{)})$
  - output is a permutation of the input: for all lists l, $l \leftrightsquigarrow \texttt{(sort } l\texttt{)}$
- How to prove programs correct?

Proving Programs Correct

- How to state properties of programs?
  - output is ordered: for all lists l, $Sorted(\texttt{(sort } l\texttt{)})$
  - output is a permutation of the input: for all lists l, $l \leftrightsquigarrow \texttt{(sort } l\texttt{)}$

- How to prove programs correct?

- **What is a correctness proof?**

Correctness Proof is Not Just About Algorithms!

```
struct Node { int data;
              Node* next; };

void insert(Node* head, int data) {
    while (head->next != nullptr)
        head = head->next;
    head->next = new Node{data, nullptr};
}
```

Correctness Proof is Not Just About Algorithms!

```
struct Node { int data;
              Node* next; };

void insert(Node* head, int data) {
    while (head->next != nullptr)
        head = head->next;
    head->next = new Node{data, nullptr};
}

Node *rest = new Node{10, nullptr};
Node *A = new Node{1, rest};

insert(A, 99);
```

Correctness Proof is Not Just About Algorithms!

```
struct Node { int data;
              Node* next; };

void insert(Node* head, int data) {
    while (head->next != nullptr)
        head = head->next;
    head->next = new Node{data, nullptr};
}

Node *rest = new Node{10, nullptr};
Node *A = new Node{1, rest};
Node *B = new Node{2, rest};
insert(A, 99);
// breaks statements about B!
```

Correctness Proof is Not Just About Algorithms!

- A model of programming languages, e.g. how program runs

Correctness Proof is Not Just About Algorithms!

- A model of programming languages, e.g. how program runs

- Powerful tools for expressing properties & making deductions

Correctness Proof is Not Just About Algorithms!

- A model of programming languages, e.g. how program runs

- Powerful tools for expressing properties & making deductions
  Separation Logic: "These two pieces of programs shall share no memory"

Correctness Proof is Not Just About Algorithms!

- A model of programming languages, e.g. how program runs
- Powerful tools for expressing properties & making deductions
  Separation Logic: “These two pieces of programs shall share no memory”
- Can the proof *guide* the implementation of programs?

Correctness Proof is Not Just About Algorithms!

- A model of programming languages, e.g. how program runs

- Powerful tools for expressing properties & making deductions
  Separation Logic: "These two pieces of programs shall share no memory"

- Can the proof *guide* the implementation of programs?

For now, we restrict our attention to a tiny subset of Racket.

Our Goal: Correctness of 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)))]))
```

Example Properties Involving Lists

Example. The `length` function distributes over `append`:
`(length (append` l l_1`))` $=$ `(length` l`)` $+$ `(length` l_1`)`.

```
> (length (append (cons 5 (cons 2 '()))
                  (cons 9 '())))
3
```

```
> (+ (length (cons 5 (cons 2 '())))
     (length (cons 9 '())))
3
```

The Data Definition of Lists

A list l is either:

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

The Data Definition of Lists

A list l is either:

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

`(cons 5 (cons 2 '()))` is a list because:

The Data Definition of Lists

A list l is either:

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

`(cons 5 (cons 2 '()))` is a list because:

- `(cons 5 (cons 2 '()))` looks like `(cons` y l'`)` where l' is `(cons 2 '())`
- `(cons 2 '())` is a (another) list because:

The Data Definition of Lists

A list l is either:

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

`(cons 5 (cons 2 '()))` is a list because:

- `(cons 5 (cons 2 '()))` looks like `(cons` y l'`)` where l' is `(cons 2 '())`
- `(cons 2 '())` is a (another) list because:
  - `(cons 2 '())` looks like `(cons` z l''`)` where l'' is `'()`
  - `'()` is a list

A Template of Induction Over Lists

Example. The `length` function distributes over `append`:
$\texttt{(length (append } l\ l_1 \texttt{))} = \texttt{(length } l \texttt{)} + \texttt{(length } l_1 \texttt{)}$.

A Template of Induction Over Lists

Example. The `length` function distributes over `append`:
$(\texttt{length}\ (\texttt{append}\ l\ l_1)) = (\texttt{length}\ l) + (\texttt{length}\ l_1)$.

Proof Template. (This is not a complete proof.)

A Template of Induction Over Lists

Example. The `length` function distributes over `append`:
`(length (append` l l_1`))` $=$ `(length` l`)` $+$ `(length` l_1`)`.

Proof Template. (This is not a complete proof.)

- Case l is `'()`: show that

  `(length (append '()` l_1`))` $=$ `(length '())` $+$ `(length` l_1`)`.

A Template of Induction Over Lists

Example. The `length` function distributes over `append`:
`(length (append` l l_1`))` $=$ `(length` l`)` $+$ `(length` l_1`)`.

Proof Template. (This is not a complete proof.)

- Case l is `'()`: show that

  `(length (append '()` l_1`))` $=$ `(length '())` $+$ `(length` l_1`)`.

- Case l is `(cons` y l'`)`: assuming that for any l'',

  `(length (append` l' l''`))` $=$ `(length` l'`)` $+$ `(length` l''`)`,

A Template of Induction Over Lists

Example. The `length` function distributes over `append`:
`(length (append` l l_1`))` $=$ `(length` l`)` $+$ `(length` l_1`)`.

Proof Template. (This is not a complete proof.)

- Case l is `'()`: show that

$$\texttt{(length (append '()}\ l_1\texttt{))} = \texttt{(length '())} + \texttt{(length}\ l_1\texttt{)}.$$

- Case l is `(cons` y l'`)`: assuming that for any l'',

$$\texttt{(length (append}\ l'\ l''\texttt{))} = \texttt{(length}\ l'\texttt{)} + \texttt{(length}\ l''\texttt{)},$$

  we need to show that

$$\texttt{(length (append (cons}\ y\ l'\texttt{)}\ l_1\texttt{))} = \texttt{(length (cons}\ y\ l'\texttt{))} + \texttt{(length}\ l_1\texttt{)}.$$

A Template of Induction Over Lists

Example. The `length` function distributes over `append`:
`(length (append` l l_1`))` $=$ `(length` l`)` $+$ `(length` l_1`)`.

Proof Template. (This is not a complete proof.)

- Case l is `'()`: show that

  `(length (append '()` l_1`))` $=$ `(length '())` $+$ `(length` l_1`)`.

- Case l is `(cons` y l'`)`: assuming that for any l'',

  `(length (append` l' l''`))` $=$ `(length` l'`)` $+$ `(length` l''`)`,

  we need to show that

  `(length (append (cons` y l'`)` l_1`))` $=$ `(length (cons` y l'`))` $+$ `(length` l_1`)`.

- By induction, `(length (append` l l_1`))` $=$ `(length` l`)` $+$ `(length` l_1`)` holds for all lists l and l_1.

Proving Properties of List Functions by Induction

Example. The `length` function distributes over `append`:
`(length (append` l l_1`))` $=$ `(length` l`)` $+$ `(length` l_1`)`.

Proof (1/4). Induction on l.

- Case l is `'()`: we need to show that
  `(length (append '()` l_1`))` $=$ `(length '())` $+$ `(length` l_1`)`.

Proving Properties of List Functions by Induction

Example. The `length` function distributes over `append`:
`(length (append` l l_1`))` $=$ `(length` l`)` $+$ `(length` l_1`)`.

Proof (1/4). Induction on l.

- Case l is `'()`: we need to show that
  `(length (append '()` l_1`))` $=$ `(length '())` $+$ `(length` l_1`)`.

  We are stuck: can't make progress with `(length (append '()` l_1`))` and `(length '())`.

"Running" Programs in Math

We will assume a programming language that

- Uses only lists, `if`, `match`, functions, number & arithmetic
- Does not use mutable variables
- All expression terminates

This way, we can partition programs into sets that "behave the same". For example, `(if #t 5 3)` should be the same as `5`.

Let $e_1 \equiv e_2$ means that the programs e_1 and e_2 are equivalent.

“Running” Programs in Math

In the end, we want to be able to deduce that:

- `(append (cons` 1 `(cons` 2 `'())) (cons` 3 `(cons` 4 `'())))` $\equiv$ `(cons` 1 `(cons` 2 `(cons` 3 `(cons` 4 `'()))))`

- `(length (cons` 3 `(cons` 4 `'())))` $\equiv 2$

- `(length (append '()` l_1`))` $\equiv$ `(length` l_1`)`

and more.

We will bake some program execution rules into “≡”.

Rules for “Running” Functions

```
(define (append xs ys)
  (match xs
    ['() ys]
    [(cons hd tl) (cons hd (append tl ys))]))
```

Rules for "Running" Functions

```
(define (append xs ys)
  (match xs
    ['() ys]
    [(cons hd tl) (cons hd (append tl ys))]))
```

We can replace `(append '()` l_1`)` by

`(append '()` l_1`)`

≡

`(match '()`
`['()` l_1`]`
`[(cons hd tl) (cons hd (append tl` l_1`))])`

Rules for “Running” Functions

```
(define (append xs ys)
  (match xs
    ['() ys]
    [(cons hd tl) (cons hd (append tl ys))]))
```

Similarly, we can replace `(append` $\texttt{(cons } y\ l'\texttt{)}$ l_1`)` by

$$
\begin{array}{l}
\quad \texttt{(append (cons } y\ l'\texttt{) } l_1\texttt{)} \\
\equiv \\
\quad \texttt{(match (cons } y\ l'\texttt{)} \\
\qquad \texttt{['()} \qquad\qquad\quad l_1\texttt{]} \\
\qquad \texttt{[(cons hd tl) (cons hd (append tl } l_1\texttt{))])}
\end{array}
$$

Rules of “Running” Matches (1)

```
(match '()
  ['() e₁]                    ≡    e₁
  [(cons hd tl) e₂])
```

Rules of “Running” Matches (1)

```
(match '()
  ['() e₁]                    ≡    e₁
  [(cons hd tl) e₂])
```

$$
\texttt{(match '()}\ \ \texttt{['()}\ e_1\texttt{]}\ \ \texttt{[(cons hd tl)}\ e_2\texttt{])} \equiv e_1
$$

Example:

$$
\texttt{(match '()}\ \ \texttt{['()}\ l_1\texttt{]}\ \ \texttt{[(cons hd tl) (cons hd (append tl}\ l_1\texttt{))])}
$$

$\equiv$

l_1

Rules of “Running” Matches (2)

$$\texttt{(match (cons } y\ l'\texttt{)} \quad \texttt{['() } e_1\texttt{]} \quad \texttt{[(cons hd tl) } e_2\texttt{])} \quad \equiv \quad e_2\{\texttt{hd} \leftarrow y, \texttt{tl} \leftarrow l'\}$$

Rules of “Running” Matches (2)

$$\begin{array}{l}\texttt{(match (cons } y\ l'\texttt{)} \\ \quad \texttt{['() } e_1\texttt{]} \\ \quad \texttt{[(cons hd tl) } e_2\texttt{])}\end{array} \quad \equiv \quad e_2\{\texttt{hd} \leftarrow y, \texttt{tl} \leftarrow l'\}$$

Example:

$$\begin{array}{l}\quad \texttt{(match (cons } y\ l'\texttt{)} \\ \quad\quad \texttt{['()} \qquad\qquad l_1\texttt{]} \\ \quad\quad \texttt{[(cons hd tl) (cons hd (append tl } l_1\texttt{))])} \\ \equiv \\ \quad \texttt{(cons } y \texttt{ (append } l'\ l_1\texttt{))}\end{array}$$

Proving Properties of List Functions by Induction

Example. The `length` function distributes over `append`:
`(length (append` l l_1`))` = `(length` l`)` + `(length` l_1`)`.

Proof (1/4). Induction on l.

- Case l is `'()`: we need to show that
  `(length (append '()` l_1`))` = `(length '())` + `(length` l_1`)`.

  By calculation in earlier slides,

  `(length (append '()` l_1`))`
  $\equiv$ `(length` l_1`)`

Proving Properties of List Functions by Induction

Example. The `length` function distributes over `append`:
`(length (append` l l_1`))` $=$ `(length` l`)` $+$ `(length` l_1`)`.

Proof (1/4). Induction on l.

- Case l is `'()`: we need to show that
  `(length (append '()` l_1`))` $=$ `(length '())` $+$ `(length` l_1`)`.

  By calculation in earlier slides,

$$
\begin{aligned}
& \texttt{(length (append '()}\ l_1\texttt{))} \\
\equiv\ & \texttt{(length}\ l_1\texttt{)} \\
\equiv\ & \texttt{(length '())} + \texttt{(length}\ l_1\texttt{)}
\end{aligned}
$$

Running the Length function

```
(define (length xs)
  (match xs
    ['() 0]
    [(cons hd tl) (+ 1 (length tl))]))
```

Running the Length function

```
(define (length xs)
  (match xs
    ['() 0]
    [(cons hd tl) (+ 1 (length tl))]))
```

We calculate:

```
    (length '())
≡
    (match '()
      ['() 0]
      [(cons hd tl) (+ 1 (length tl))])
```

Running the Length function

```
(define (length xs)
  (match xs
    ['() 0]
    [(cons hd tl) (+ 1 (length tl))]))
```

We calculate:

```
    (length '())
≡
    (match '()
      ['() 0]
      [(cons hd tl) (+ 1 (length tl))])
≡
    0
```

Proving Properties of List Functions by Induction (cont'd)

Example. The `length` function distributes over `append`:
`(length (append` l l_1`))` = `(length` l`)` + `(length` l_1`)`.

Proof (1/4). Induction on l.

- Case l is `'()`: we need to show that
  `(length (append '()` l_1`))` = `(length '())` + `(length` l_1`)`.

  By calculation in earlier slides,

$$
\begin{aligned}
& \texttt{(length (append '()}\ l_1\texttt{))} \\
\equiv\ & \texttt{(length}\ l_1\texttt{)} \\
\equiv\ & \texttt{(length '())} + \texttt{(length}\ l_1\texttt{)}
\end{aligned}
$$

Proving Properties of List Functions by Induction (cont'd)

Example. The `length` function distributes over `append`:
`(length (append` l l_1`))` = `(length` l`)` + `(length` l_1`)`.

Proof (2/4). Induction on l.

- Case l is `'()`: we need to show that
  `(length (append '()` l_1`))` = `(length '())` + `(length` l_1`)`.

  By calculation in earlier slides,

$$
\begin{array}{rc}
 & \texttt{(length (append '()}\ l_1\texttt{))} \\
\equiv & \texttt{(length}\ l_1\texttt{)} \\
= & 0 + \texttt{(length}\ l_1\texttt{)} \\
\equiv & \texttt{(length '())} + \texttt{(length}\ l_1\texttt{)}
\end{array}
$$

Recap: Proving Properties of List Functions by Induction

Example. The `length` function distributes over `append`:
$\texttt{(length (append } l\ l_1 \texttt{))} = \texttt{(length } l \texttt{)} + \texttt{(length } l_1 \texttt{)}$.

Proof Template.

- ✓ Case l is `'()`: show that

  $$\texttt{(length (append '() } l_1 \texttt{))} = \texttt{(length '())} + \texttt{(length } l_1 \texttt{)}.$$

- **TODO** Case l is $\texttt{(cons } y\ l' \texttt{)}$: assuming that for any l'',
  $\texttt{(length (append } l'\ l'' \texttt{))} = \texttt{(length } l' \texttt{)} + \texttt{(length } l'' \texttt{)}$, we need to show

  $$\texttt{(length (append (cons } y\ l' \texttt{) } l_1 \texttt{))} = \texttt{(length (cons } y\ l' \texttt{))} + \texttt{(length } l_1 \texttt{)}.$$

- By induction, $\texttt{(length (append } l\ l_1 \texttt{))} = \texttt{(length } l \texttt{)} + \texttt{(length } l_1 \texttt{)}$ holds for all lists l and l_1.

Proving Properties of List Functions by Induction (cont'd)

Example. The `length` function distributes over `append`:
$(\texttt{length}\ (\texttt{append}\ l\ l_1)) = (\texttt{length}\ l) + (\texttt{length}\ l_1)$.

Proof (3/4).

- Case l is $(\texttt{cons}\ y\ l')$: we need to show that if for any l'', $(\texttt{length}\ (\texttt{append}\ l'\ l'')) = (\texttt{length}\ l') + (\texttt{length}\ l'')$ then we have $(\texttt{length}\ (\texttt{append}\ (\texttt{cons}\ y\ l')\ l_1)) = (\texttt{length}\ (\texttt{cons}\ y\ l')) + (\texttt{length}\ l_1)$:

More Calculation (1)

```
  (length (append (cons y l′) l₁))
≡         (the rule of function call)
  (length (match (cons y l′)
               ['() l₁]
               [(cons hd tl) (cons hd (append tl l₁))]))
```

```
(define (append xs ys)
  (match xs
    ['() ys]
    [(cons hd tl) (cons hd (append tl ys))]))
```

More Calculation (1)

```
   (length (append (cons y l') l1))
≡          (the rule of function call)
   (length (match (cons y l')
                  ['() l1]
                  [(cons hd tl) (cons hd (append tl l1))]))
≡          (the rules of match)
   (length (cons y (append l' l1)))
```

```
(define (length xs)
  (match xs
    ['() 0]
    [(cons hd tl) (+ 1 (length tl))]))
```

More Calculation (2)

```
  (length (append (cons y l′) l₁))
≡         (the rule of function call)
  (length (match (cons y l′)
                 ['() l₁]
                 [(cons hd tl) (cons hd (append tl l₁))]))
≡         (the rules of match)
  (length (cons y (append l′ l₁)))
≡         (the rule of function call)
  (match (cons y (append l′ l₁))
    ['() 0]
    [(cons hd tl) (+ 1 (length tl))])
```

More Calculation (2)

$$
\begin{array}{ll}
 & \texttt{(length (append } \color{blue}{\texttt{(cons } y\ l'\texttt{)}}\ l_1\texttt{))} \\
\equiv & \qquad \color{green}{\textit{(the rule of function call)}} \\
 & \texttt{(length (match } \color{blue}{\texttt{(cons } y\ l'\texttt{)}} \\
 & \qquad\qquad \texttt{['() } l_1\texttt{]} \\
 & \qquad\qquad \texttt{[(cons hd tl) (cons hd (append tl } l_1\texttt{))]))} \\
\equiv & \qquad \color{green}{\textit{(the rules of } \texttt{match}\textit{)}} \\
 & \texttt{(length (cons } \color{blue}{y} \texttt{ (append } \color{blue}{l'}\ l_1\texttt{)))} \\
\equiv & \qquad \color{green}{\textit{(the rule of function call)}} \\
 & \texttt{(match (cons } \color{blue}{y} \texttt{ (append } \color{blue}{l'}\ l_1\texttt{))} \\
 & \quad \texttt{['() 0]} \\
 & \quad \texttt{[(cons hd tl) (+ 1 (length tl))])} \\
\equiv & \qquad \color{green}{\textit{(the rules of } \texttt{match}\textit{)}} \\
 & \texttt{(+ 1 (length (append } \color{blue}{l'}\ l_1\texttt{)))}
\end{array}
$$

Proving Properties of List Functions by Induction (cont'd)

Example. The `length` function distributes over `append`:
$(\texttt{length}\ (\texttt{append}\ l\ l_1)) = (\texttt{length}\ l) + (\texttt{length}\ l_1)$.

Proof (3/4).

- Case l is $(\texttt{cons}\ y\ l')$: we need to show that if for any l'', $(\texttt{length}\ (\texttt{append}\ l'\ l'')) = (\texttt{length}\ l') + (\texttt{length}\ l'')$ then we have $(\texttt{length}\ (\texttt{append}\ (\texttt{cons}\ y\ l')\ l_1)) = (\texttt{length}\ (\texttt{cons}\ y\ l')) + (\texttt{length}\ l_1)$:

$$
\begin{aligned}
(\texttt{length}\ (\texttt{append}\ (\texttt{cons}\ y\ l')\ l_1)) &\equiv (\texttt{length}\ (\texttt{cons}\ y\ (\texttt{append}\ l'\ l_1))) \\
&\equiv 1 + (\texttt{length}\ (\texttt{append}\ l'\ l_1))
\end{aligned}
$$

Proving Properties of List Functions by Induction (cont'd)

Example. The `length` function distributes over `append`:
$(\texttt{length}\,(\texttt{append}\ l\ l_1)) = (\texttt{length}\ l) + (\texttt{length}\ l_1)$.

Proof (3/4).

- Case l is $(\texttt{cons}\ y\ l')$: we need to show that if for any l'', $(\texttt{length}\,(\texttt{append}\ l'\ l'')) = (\texttt{length}\ l') + (\texttt{length}\ l'')$ then we have $(\texttt{length}\,(\texttt{append}\,(\texttt{cons}\ y\ l')\ l_1)) = (\texttt{length}\,(\texttt{cons}\ y\ l')) + (\texttt{length}\ l_1)$:

$$
\begin{aligned}
(\texttt{length}\,(\texttt{append}\,(\texttt{cons}\ y\ l')\ l_1)) &\equiv (\texttt{length}\,(\texttt{cons}\ y\,(\texttt{append}\ l'\ l_1))) \\
&\equiv 1 + (\texttt{length}\,(\texttt{append}\ l'\ l_1)) \\
& \\
&\ 1 + (\texttt{length}\ l') + (\texttt{length}\ l_1) \\
&\equiv (\texttt{length}\,(\texttt{cons}\ y\ l')) + (\texttt{length}\ l_1)
\end{aligned}
$$

Proving Properties of List Functions by Induction (cont'd)

Example. The `length` function distributes over `append`:
$(\texttt{length}\,(\texttt{append}\ l\ l_1)) = (\texttt{length}\ l) + (\texttt{length}\ l_1)$.

Proof (4/4).

- Case l is $(\texttt{cons}\ y\ l')$: we need to show that if for any l'', $(\texttt{length}\,(\texttt{append}\ l'\ l'')) = (\texttt{length}\ l') + (\texttt{length}\ l'')$ then we have $(\texttt{length}\,(\texttt{append}\,(\texttt{cons}\ y\ l')\ l_1)) = (\texttt{length}\,(\texttt{cons}\ y\ l')) + (\texttt{length}\ l_1)$:

$$\begin{aligned}
(\texttt{length}\,(\texttt{append}\,(\texttt{cons}\ y\ l')\ l_1)) &\equiv (\texttt{length}\,(\texttt{cons}\ y\,(\texttt{append}\ l'\ l_1))) \\
&\equiv 1 + (\texttt{length}\,(\texttt{append}\ l'\ l_1)) \\
&\equiv \textit{(induction hypothesis)} \\
&\ 1 + (\texttt{length}\ l') + (\texttt{length}\ l_1) \\
&\equiv (\texttt{length}\,(\texttt{cons}\ y\ l')) + (\texttt{length}\ l_1)
\end{aligned}$$

Proving Properties of List Functions by Induction (cont'd)

Example. The `length` function distributes over `append`:
$(\texttt{length}\ (\texttt{append}\ l\ l_1)) = (\texttt{length}\ l) + (\texttt{length}\ l_1)$.

Proof (4/4).

- Case l is $(\texttt{cons}\ y\ l')$: we need to show that if for any l'', $(\texttt{length}\ (\texttt{append}\ l'\ l'')) = (\texttt{length}\ l') + (\texttt{length}\ l'')$ then we have $(\texttt{length}\ (\texttt{append}\ (\texttt{cons}\ y\ l')\ l_1)) = (\texttt{length}\ (\texttt{cons}\ y\ l')) + (\texttt{length}\ l_1)$:

$$
\begin{aligned}
(\texttt{length}\ (\texttt{append}\ (\texttt{cons}\ y\ l')\ l_1)) &\equiv (\texttt{length}\ (\texttt{cons}\ y\ (\texttt{append}\ l'\ l_1))) \\
&\equiv 1 + (\texttt{length}\ (\texttt{append}\ l'\ l_1)) \\
&\equiv \textit{(induction hypothesis)} \\
&\ 1 + (\texttt{length}\ l') + (\texttt{length}\ l_1) \\
&\equiv (\texttt{length}\ (\texttt{cons}\ y\ l')) + (\texttt{length}\ l_1)
\end{aligned}
$$

By induction, $(\texttt{length}\ (\texttt{append}\ l\ l_1)) \equiv (\texttt{length}\ l) + (\texttt{length}\ l_1)$.

Sum Up: Proving Properties of List Functions by Induction

Property: "For all lists l, ... l ..."

Proof (template).

Induction on l:

- Case l is `'()`: ... `'()` ...
- Case l is `(cons` y l'`)`: if ... l' ... then ... `(cons` y l'`)` ....

By induction, ... l ....

How Induction "Runs"

Example. The `length` function distributes over append:
`(length (append` l l_1`))` $=$ `(length` l`)` $+$ `(length` l_1`)`.

1. `(length (append '() (cons 9 '())))` $=$
   `(length '())` $+$ `(length (cons 9 '()))`

We have shown that `(length (append '()` l_1`))` $=$ `(length '())` $+$ `(length` l_1`)`.
In this specific instance, l_1 is `(cons 9 '())`.

How Induction "Runs"

Example. The `length` function distributes over append:
`(length (append` l l_1`))` $=$ `(length` l`)` $+$ `(length` l_1`)`.

1. `(length (append '() (cons 9 '())))` $=$
   `(length '())` $+$ `(length (cons 9 '()))`
2. `(length (append (cons 2 '()) (cons 9 '())))` $=$
   `(length (cons 2 '()))` $+$ `(length (cons 9 '()))`

We have shown that if for any l'',
`(length (append` l' l''`))` $=$ `(length` l'`)` $+$ `(length` l''`)` then we have
`(length (append (cons` y l'`)` l_1`))` $=$ `(length (cons` y l'`))` $+$ `(length` l_1`)`.
Here the premise is true by (1), $y := 2$ and $l' :=$ `'()`.

How Induction "Runs"

Example. The `length` function distributes over `append`:
$(\texttt{length}\ (\texttt{append}\ l\ l_1)) = (\texttt{length}\ l) + (\texttt{length}\ l_1)$.

1. $(\texttt{length}\ (\texttt{append}\ \texttt{'()}\ (\texttt{cons}\ 9\ \texttt{'()}))) =$
   $(\texttt{length}\ \texttt{'()}) + (\texttt{length}\ (\texttt{cons}\ 9\ \texttt{'()}))$
2. $(\texttt{length}\ (\texttt{append}\ (\texttt{cons}\ 2\ \texttt{'()})\ (\texttt{cons}\ 9\ \texttt{'()}))) =$
   $(\texttt{length}\ (\texttt{cons}\ 2\ \texttt{'()})) + (\texttt{length}\ (\texttt{cons}\ 9\ \texttt{'()}))$
3. $(\texttt{length}\ (\texttt{append}\ (\texttt{cons}\ 5\ (\texttt{cons}\ 2\ \texttt{'()}))\ (\texttt{cons}\ 9\ \texttt{'()}))) =$
   $(\texttt{length}\ (\texttt{cons}\ 5\ (\texttt{cons}\ 2\ \texttt{'()}))) + (\texttt{length}\ (\texttt{cons}\ 9\ \texttt{'()}))$

We have shown that if for any l'',
$(\texttt{length}\ (\texttt{append}\ l'\ l'')) = (\texttt{length}\ l') + (\texttt{length}\ l'')$ then we have
$(\texttt{length}\ (\texttt{append}\ (\texttt{cons}\ y\ l')\ l_1)) = (\texttt{length}\ (\texttt{cons}\ y\ l')) + (\texttt{length}\ l_1)$.
Here the premise is true by (2), $y := 5$ and $l' := (\texttt{cons}\ 2\ \texttt{'()})$.

Appendix: Rules of Function Calls

For any function definition

```
(define (f x1 x2 ...)
  e)
```

We have the computation rule

$$(f\ e_1\ e_2\ \ldots) \qquad \equiv \qquad e\{x_1 \leftarrow e_1,\ \ x_2 \leftarrow e_2,\ \ \ldots\}$$

Appendix: Rules of Match

$$\texttt{(match '()}\ \ \texttt{['()}\ e_1\texttt{]}\ \ \texttt{[(cons hd tl)}\ e_2\texttt{])} \equiv e_1$$

$$\texttt{(match (cons}\ x\ l\texttt{)}\ \ \texttt{['()}\ e_1\texttt{]}\ \ \texttt{[(cons hd tl)}\ e_2\texttt{])} \equiv e_2\{\texttt{hd} \leftarrow x, \texttt{tl} \leftarrow l\}$$

Appendix: Rules of If

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

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