COMP2111 Week 2
Term 1, 2019
Recursion and induction

Summary of topics

- Recursion
- Recursive Data Types
- Induction
- Structural Induction

Summary of topics

- Recursion
- Recursive Data Types
- Induction
- Structural Induction

Recursion

Fundamental concept in Computer Science

- Recursion in algorithms: Solving problems by reducing to smaller cases
  - Factorial
  - Towers of Hanoi
  - Mergesort, Quicksort
- Recursion in data structures: Finite definitions of **arbitrarily large** objects
  - Natural numbers
  - Words
  - Linked lists
  - Formulas
  - Binary trees
- Analysis of recursion: Proving properties
  - Recursive sequences (e.g. Fibonacci sequence)
  - Structural induction

Recursion

Fundamental concept in Computer Science

- Recursion in algorithms: Solving problems by reducing to smaller cases
  - Factorial
  - Towers of Hanoi
  - Mergesort, Quicksort
- Recursion in data structures: Finite definitions of **arbitrarily large** objects
  - Natural numbers
  - Words
  - Linked lists
  - Formulas
  - Binary trees
- Analysis of recursion: Proving properties
  - Recursive sequences (e.g. Fibonacci sequence)
  - Structural induction

Recursion

Fundamental concept in Computer Science

- Recursion in algorithms: Solving problems by reducing to smaller cases
  - Factorial
  - Towers of Hanoi
  - Mergesort, Quicksort
- Recursion in data structures: Finite definitions of **arbitrarily large** objects
  - Natural numbers
  - Words
  - Linked lists
  - Formulas
  - Binary trees
- Analysis of recursion: Proving properties
  - Recursive sequences (e.g. Fibonacci sequence)
  - Structural induction

Recursion

Consists of a basis (B) and recursive process (R).
A sequence/object/algorithm is recursively defined when (typically)
(B) some initial terms are specified, perhaps only the first one;
(R) later terms stated as functional expressions of the earlier terms.

NB

(R) also called **recurrence formula (especially when dealing with sequences)**

Example: Factorial

Factorial:

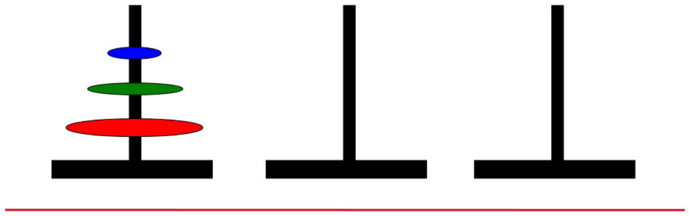
(B)	$0! = 1$
(R)	$(n+1)! = (n+1) \cdot n!$

```
         fact(n):
(B)         if(n = 0): 1
(R)         else: n * fact(n − 1)
```

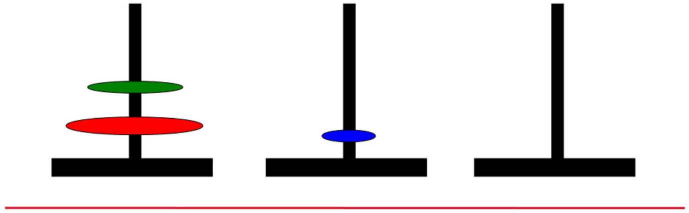
Example: Towers of Hanoi

- There are 3 towers (pegs)
- n disks of decreasing size placed on the first tower
- You need to move all disks from the first tower to the last tower
- Larger disks cannot be placed on top of smaller disks
- The third tower can be used to temporarily hold disks

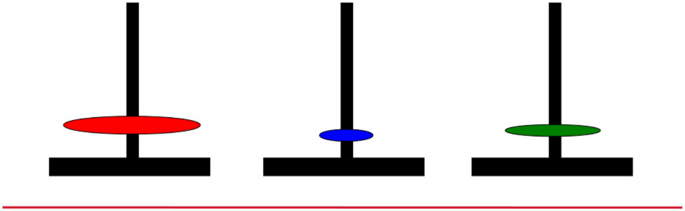
Example: Towers of Hanoi



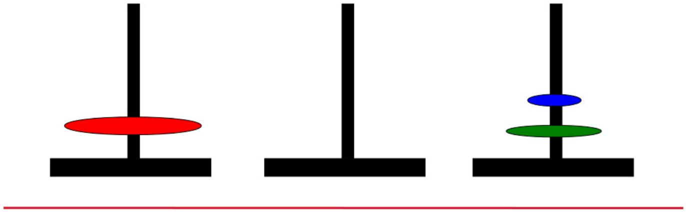
Example: Towers of Hanoi



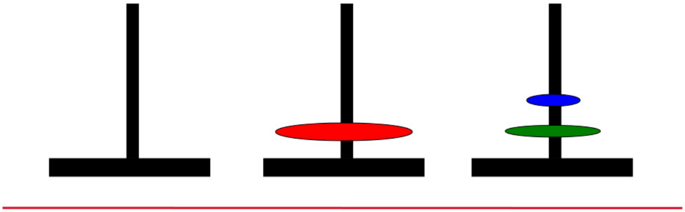
Example: Towers of Hanoi



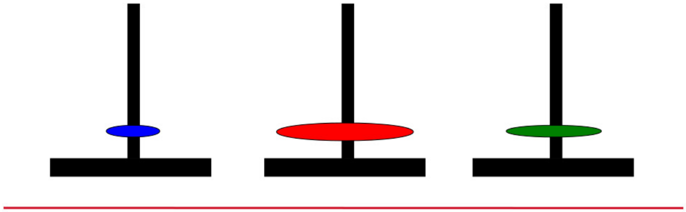
Example: Towers of Hanoi



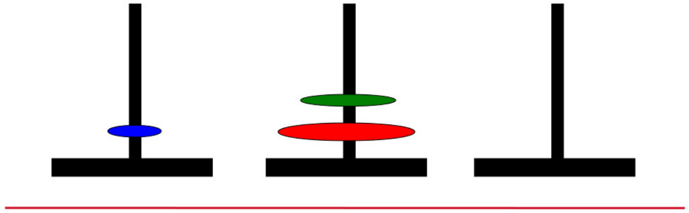
Example: Towers of Hanoi



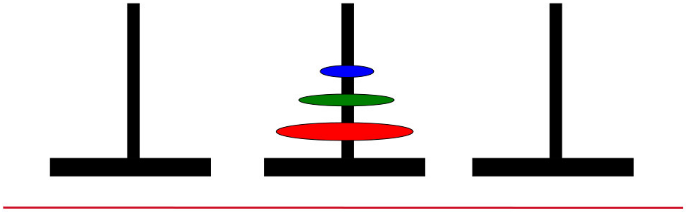
Example: Towers of Hanoi



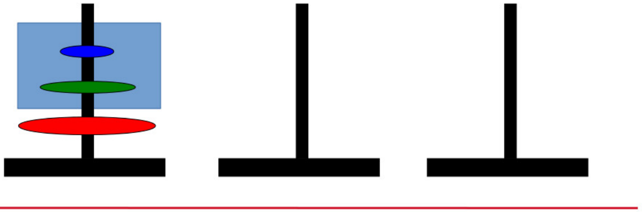
Example: Towers of Hanoi



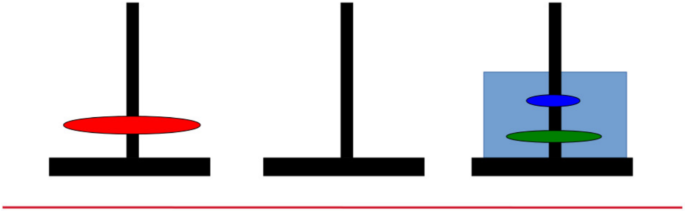
Example: Towers of Hanoi



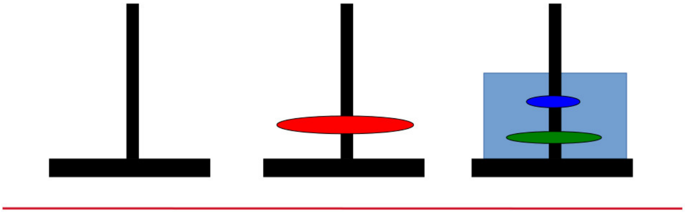
Example: Towers of Hanoi



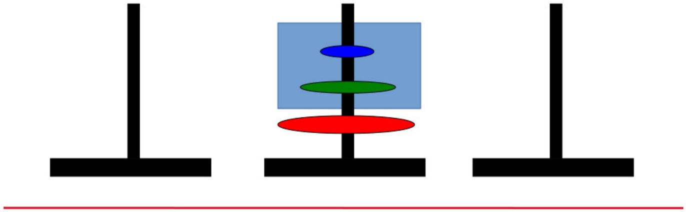
Example: Towers of Hanoi



Example: Towers of Hanoi



Example: Towers of Hanoi



Summary of topics

- Recursion
- Recursive Data Types
- Induction
- Structural Induction

Example: Natural numbers

A natural number is either 0 (B) or one more than a natural number (R).

Formal definition of $\mathbb{N}$:

- (B) $0 \in \mathbb{N}$
- (R) If $n \in \mathbb{N}$ then $(n + 1) \in \mathbb{N}$

Example: Fibonacci numbers

The Fibonacci sequence starts $0, 1, 1, 2, 3, \ldots$ where, after $0, 1$, each term is the sum of the previous two terms.

Formally, the set of Fibonacci numbers: $\mathbb{F} = \{F_n : n \in \mathbb{N}\}$, where the n-th Fibonacci number F_n is defined as:

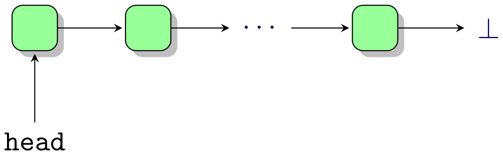
- (B) $F_0 = 0$,
- (B) $F_1 = 1$,
- (I) $F_n = F_{n-1} + F_{n-2}$

NB

Could also define the Fibonacci sequence as a function $\textsc{fib} : \mathbb{N} \to \mathbb{F}$. *Choice of perspective depends on what structure you view as your base object (***ground type***).*

Example: Linked lists

Recall: A linked list is zero or more linked list nodes:

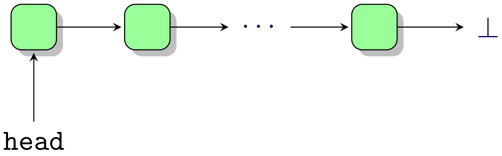


In C:

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

Example: Linked lists

Recall: A linked list is zero or more linked list nodes:



In C:

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

Example: Linked lists

We can view the linked list **structure** abstractly. A linked list is either:

- (B) an empty list, or
- (R) an ordered pair $(\texttt{Data}, \mathsf{List})$.

Example: Words over Σ

A word over an alphabet Σ is either λ (B) or a symbol from Σ followed by a word (R).

Formal definition of Σ^*:

- (B) $\lambda \in \Sigma^*$
- (R) If $w \in \Sigma^*$ then $aw \in \Sigma^*$ for all $a \in \Sigma$

NB

This matches the recursive definition of a **Linked List** *data type.*

Example: Propositional formulas

A well-formed formula (wff) over a set of propositional variables, $\textsc{Prop}$ is defined as:

- (B) $\top$ is a wff
- (B) $\bot$ is a wff
- (B) p is a wff for all $p \in \textsc{Prop}$
- (R) If φ is a wff then $\neg\varphi$ is a wff
- (R) If φ and ψ are wffs then:
  - $(\varphi \wedge \psi)$,
  - $(\varphi \vee \psi)$,
  - $(\varphi \rightarrow \psi)$, and
  - $(\varphi \leftrightarrow \psi)$ are wffs.

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

The factorial function:

```
      fact($n$):
(B)     if($n = 0$): 1
(R)     else: $n * $fact($n - 1$)
```

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

Summing the first n natural numbers:

```
        sum(n):
(B)        if(n = 0): 0
(R)        else: n + sum(n − 1)
```

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

Summing elements of a linked list: (see tutorial)

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

Concatenation of words (defining wv):

$$\begin{array}{ll} & \text{For all } w, v \in \Sigma^* \text{ and } a \in \Sigma : \\ (B) & \lambda v = v \\ (R) & (aw)v = a(wv) \end{array}$$

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

Length of words:

$$\begin{array}{ll} (B) & \mathrm{length}(\lambda) = 0 \\ (R) & \mathrm{length}(aw) = 1 + \mathrm{length}(w) \end{array}$$

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

"Evaluation" of a propositional formula

Summary of topics

- Recursion
- Recursive Data Types
- Induction
- Structural Induction

Recursive datatypes
: Describe arbitrarily large objects in a finite way

Recursive functions
: Define behaviour for these objects in a finite way

Induction
: Reason about these objects in a finite way

Inductive Reasoning

Suppose we would like to reach a conclusion of the form

$$P(x) \textit{ for all } x \text{ (of some type)}$$

Inductive reasoning (as understood in philosophy) proceeds from examples.

E.g. From "This swan is white, that swan is white, in fact every swan I have seen so far is white"

Conclude: "Every Swan is white"

NB

This may be a good way to discover hypotheses.
But it is not a valid principle of reasoning!

Mathematical induction *is a variant that is valid.*

Inductive Reasoning

Suppose we would like to reach a conclusion of the form

$$P(x) \textit{ for all } x \text{ (of some type)}$$

Inductive reasoning (as understood in philosophy) proceeds from examples.
E.g. From "This swan is white, that swan is white, in fact every swan I have seen so far is white"
Conclude: "Every Swan is white"

NB

This may be a good way to discover hypotheses.
But it is not a valid principle of reasoning!

Mathematical induction *is a variant that is valid.*

Mathematical Induction

Mathematical Induction is based not just on a set of examples, but also a rule for deriving new cases of $P(x)$ from cases for which P is known to hold.
General structure of reasoning by mathematical induction:

Base Case [B]: $P(a_1), P(a_2), \ldots, P(a_n)$ for some small set of examples $a_1 \ldots a_n$ (often $n = 1$)
Inductive Step [I]: A general rule showing that if $P(x)$ holds for some cases $x = x_1, \ldots, x_k$ then $P(y)$ holds for some new case y, constructed in some way from $x_1, \ldots, x_k$.

Conclusion: Starting with $a_1 \ldots a_n$ and repeatedly applying the construction of y from existing values, we can eventually construct all values in the domain of interest.

Basic induction

Basic induction is this principle applied to the natural numbers.

Goal: Show $P(n)$ holds for all $n \in \mathbb{N}$.

Approach: Show that:

Base case (B): $P(0)$ holds; and

Inductive case (I): If $P(k)$ holds then $P(k+1)$ holds.

Example

Recall the recursive program:

Example

Summing the first n natural numbers:

```
sum(n):
   if(n = 0): 0
   else: n + sum(n − 1)
```

Another attempt:

Example

```
sum2(n):
   return n ∗ (n + 1)/2
```

Induction proof **guarantees** that these programs will behave the same.

Example

Let $P(n)$ be the proposition that:

$$P(n): \qquad \sum_{i=0}^{n} i = \frac{n(n+1)}{2}.$$

We will show that $P(n)$ holds for all $n \in \mathbb{N}$ by induction on n.

Proof.

[B] $P(0)$, i.e.

$$\sum_{i=0}^{0} i = \frac{0(0+1)}{2}$$

[I] $\forall k \geq 0\,(P(k) \rightarrow P(k+1))$, i.e.

$$\sum_{i=0}^{k} i = \frac{k(k+1)}{2} \rightarrow \sum_{i=0}^{k+1} i = \frac{(k+1)(k+2)}{2}$$

(proof?) □

Example

Let $P(n)$ be the proposition that:

$$P(n): \qquad \sum_{i=0}^{n} i \ = \frac{n(n+1)}{2}.$$

We will show that $P(n)$ holds for all $n \in \mathbb{N}$ by induction on n.

Proof.

[B] $P(0)$, i.e.

$$\sum_{i=0}^{0} \ i \ = \frac{0(0+1)}{2}$$

[I] $\forall k \geq 0\,(P(k) \to P(k+1))$, i.e.

$$\sum_{i=0}^{k} i \ = \frac{k(k+1)}{2} \quad \to \quad \sum_{i=0}^{k+1} i \ = \frac{(k+1)(k+2)}{2}$$

(proof?) $\square$

Example

Let $P(n)$ be the proposition that:

$$P(n): \quad \sum_{i=0}^{n} i = \frac{n(n+1)}{2}.$$

We will show that $P(n)$ holds for all $n \in \mathbb{N}$ by induction on n.

Proof.

[B] $P(0)$, i.e.

$$\sum_{i=0}^{0} i = \frac{0(0+1)}{2}$$

[I] $\forall k \geq 0\,(P(k) \to P(k+1))$, i.e.

$$\sum_{i=0}^{k} i = \frac{k(k+1)}{2} \to \sum_{i=0}^{k+1} i = \frac{(k+1)(k+2)}{2}$$

(proof?) $\square$

Example (cont'd)

Proof.

Inductive step [I]:

$$\begin{aligned}
\sum_{i=0}^{k+1} i &= \left(\sum_{i=0}^{k} i\right) + (k+1) \\
&= \frac{k(k+1)}{2} + (k+1) \qquad \text{(by the inductive hypothesis)} \\
&= \frac{k(k+1) + 2(k+1)}{2} \\
&= \frac{(k+1)(k+2)}{2}
\end{aligned}$$

□

Variations

1. Induction from m upwards
2. Induction steps > 1
3. Strong induction
4. Backward induction
5. Forward-backward induction
6. Structural induction

Induction From m Upwards

If

[B] $P(m)$

[I] $\forall k \geq m \, (P(k) \to P(k+1))$

then

[C] $\forall n \geq m \, (P(n))$

Example

Theorem. For all $n \geq 1$, the number $8^n - 2^n$ is divisible by 6.

[B] $8^1 - 2^1$ is divisible by 6
[I] if $8^k - 2^k$ is divisible by 6, then so is $8^{k+1} - 2^{k+1}$, for all $k \geq 1$

Prove [I] using the "trick" to rewrite 8^{k+1} as $8 \cdot (8^k - 2^k + 2^k)$ which allows you to apply the IH on $8^k - 2^k$

Induction Steps $\ell > 1$

If

[B]	$P(m)$
[I]	$P(k) \to P(k+\ell)$ for all $k \geq m$

then

[C]	$P(n)$ for every ℓ'th $n \geq m$

Example

Every 4th Fibonacci number is divisible by 3.

[B] $F_4 = 3$ is divisible by 3
[I] if $3 \mid F_k$, then $3 \mid F_{k+4}$, for all $k \geq 4$

Prove [I] by rewriting F_{k+4} in such a way that you can apply the IH on F_k

Strong Induction

This is a version in which the inductive hypothesis is stronger. Rather than using the fact that $P(k)$ holds for a single value, we use *all* values up to k.

If

[B] $P(m)$

[I] $[P(m) \wedge P(m+1) \wedge \ldots \wedge P(k)] \rightarrow P(k+1)$ for all $k \geq m$

then

[C] $P(n)$, for all $n \geq m$

Example

Claim: All integers ≥ 2 can be written as a product of primes.

[B] 2 is a product of primes
[I] If all x with $2 \leq x \leq k$ can be written as a product of primes, then $k+1$ can be written as a product of primes, for all $k \geq 2$

Proof for [I]?

Negative Integers, Backward Induction

NB

Induction can be conducted over any subset of $\mathbb{Z}$ with least element. Thus m can be negative; eg. base case $m = -10^6$.

NB

One can apply induction in the 'opposite' direction $p(m) \rightarrow p(m-1)$. It means considering the integers with the opposite ordering where the next number after n is $n-1$. Such induction would be used to prove some $p(n)$ for all $n \leq m$.

NB

Sometimes one needs to reason about all integers $\mathbb{Z}$. This requires two separate simple induction proofs: one for $\mathbb{N}$, another for $-\mathbb{N}$. They both would start form some initial values, which could be the same, e.g. zero. Then the first proof would proceed through positive integers; the second proof through negative integers.

Forward-Backward Induction

Idea

To prove $P(n)$ for all $n \geq k_0$

- *verify $P(k_0)$*
- *prove $P(k_i)$ for infinitely many $k_0 < k_1 < k_2 < k_3 < \ldots$*
- *fill the gaps*
  $P(k_1) \rightarrow P(k_1 - 1) \rightarrow P(k_1 - 2) \rightarrow \ldots \rightarrow P(k_0 + 1)$
  $P(k_2) \rightarrow P(k_2 - 1) \rightarrow P(k_2 - 2) \rightarrow \ldots \rightarrow P(k_1 + 1)$
  $\ldots\ldots\ldots$

NB

This form of induction is extremely important for the analysis of algorithms.

Summary of topics

- Recursion
- Recursive Data Types
- Induction
- Structural Induction

Structural Induction

Basic induction allows us to assert properties over **all natural numbers**. The induction scheme (layout) uses the recursive definition of $\mathbb{N}$.

The induction schemes can be applied not only to natural numbers (and integers) but to any partially ordered set in general – especially those defined recursively.

The basic approach is always the same — we need to verify that

- **[B]** the property holds for all minimal objects — objects that have no predecessors; they are usually very simple objects allowing immediate verification
- **[I]** for any given object, if the property in question holds for all its predecessors ('smaller' objects) then it holds for the object itself

Example: Induction on Σ^*

Recall definition of Σ^*:

$$\lambda \in \Sigma^*$$

$$\text{If } w \in \Sigma^* \text{ then } aw \in \Sigma^* \text{ for all } a \in \Sigma$$

Structural induction on Σ^*:

Goal: Show $P(w)$ holds for all $w \in \Sigma^*$.

Approach: Show that:

Base case (B): $P(\lambda)$ holds; and

Inductive case (I): If $P(w)$ holds then $P(aw)$ holds for all $a \in \Sigma$.

Example: Induction on Σ^*

Recall:

Formal definition of Σ^*:

$\lambda \in \Sigma^*$

If $w \in \Sigma^*$ then $aw \in \Sigma^*$ for all $a \in \Sigma$

Formal definition of concatenation:

(concat.B) $\lambda v = v$

(concat.I) $(aw)v = a(wv)$

Formal definition of length:

(length.B) $\text{length}(\lambda) = 0$

(length.I) $\text{length}(aw) = 1 + \text{length}(w)$

Prove:

$$\text{length}(wv) = \text{length}(w) + \text{length}(v)$$

Example: Induction on Σ^*

Recall:

Formal definition of Σ^*:

$$\lambda \in \Sigma^*$$

If $w \in \Sigma^*$ then $aw \in \Sigma^*$ for all $a \in \Sigma$

Formal definition of concatenation:

(concat.B) $\lambda v = v$

(concat.I) $(aw)v = a(wv)$

Formal definition of length:

(length.B) $\text{length}(\lambda) = 0$

(length.I) $\text{length}(aw) = 1 + \text{length}(w)$

Prove:

$$\text{length}(wv) = \text{length}(w) + \text{length}(v)$$

Example: Induction on Σ^*

Let $P(w)$ be the proposition that, for all $v \in \Sigma^*$:

$$\text{length}(wv) = \text{length}(w) + \text{length}(v).$$

We will show that $P(w)$ holds for all $w \in \Sigma^*$ by **structural induction on w.**

Proof:

Example: Induction on Σ^*

Let $P(w)$ be the proposition that, for all $v \in \Sigma^*$:

$$\text{length}(wv) = \text{length}(w) + \text{length}(v).$$

We will show that $P(w)$ holds for all $w \in \Sigma^*$ by **structural induction on w.**

Proof:
Base case ($w = \lambda$):

$$\begin{aligned}
\text{length}(\lambda v) &= \text{length}(v) && (\text{concat.B}) \\
&= 0 + \text{length}(v) \\
&= \text{length}(w) + \text{length}(v) && (\text{length.B})
\end{aligned}$$

Example: Induction on Σ^*

Let $P(w)$ be the proposition that, for all $v \in \Sigma^*$:

$$\text{length}(wv) = \text{length}(w) + \text{length}(v).$$

We will show that $P(w)$ holds for all $w \in \Sigma^*$ by **structural induction on w.**

Proof:
Base case ($w = \lambda$):

$$\begin{aligned}
\text{length}(\lambda v) &= \text{length}(v) && \text{(concat.B)}\\
&= 0 + \text{length}(v) &&\\
&= \text{length}(w) + \text{length}(v) && \text{(length.B)}
\end{aligned}$$

Example: Induction on Σ^*

Let $P(w)$ be the proposition that, for all $v \in \Sigma^*$:

$$\text{length}(wv) = \text{length}(w) + \text{length}(v).$$

We will show that $P(w)$ holds for all $w \in \Sigma^*$ by **structural induction on w.**

Proof:
Base case ($w = \lambda$):

$$\begin{aligned} \text{length}(\lambda v) &= \text{length}(v) && \text{(concat.B)} \\ &= 0 + \text{length}(v) \\ &= \text{length}(w) + \text{length}(v) && \text{(length.B)} \end{aligned}$$

Example: Induction on Σ^*

Let $P(w)$ be the proposition that, for all $v \in \Sigma^*$:

$$\text{length}(wv) = \text{length}(w) + \text{length}(v).$$

We will show that $P(w)$ holds for all $w \in \Sigma^*$ by **structural induction on** w**.**

Proof:
Base case ($w = \lambda$**):**

$$\begin{aligned} \text{length}(\lambda v) &= \text{length}(v) && \text{(concat.B)} \\ &= 0 + \text{length}(v) && \\ &= \text{length}(w) + \text{length}(v) && \text{(length.B)} \end{aligned}$$

Example: Induction on Σ^*

Proof cont'd:

Inductive case ($w = aw'$): Assume that $P(w')$ holds. That is, for all $v \in \Sigma^*$:

$$\text{(IH)}: \qquad \text{length}(w'v) = \text{length}(w') + \text{length}(v).$$

Then, for all $a \in \Sigma$, we have:

$$\begin{aligned}
\text{length}((aw')v) &= \text{length}(a(w'v)) && \text{(concat.I)}\\
&= 1 + \text{length}(w'v) && \text{(length.I)}\\
&= 1 + \text{length}(w') + \text{length}(v) && \text{(IH)}\\
&= \text{length}(aw') + \text{length}(v) && \text{(length.I)}
\end{aligned}$$

So $P(aw')$ holds.

We have $P(\lambda)$ and for all $w' \in \Sigma^*$ and $a \in \Sigma$: $P(w') \rightarrow P(aw')$.
Hence $P(w)$ holds for all $w \in \Sigma^*$.

Example: Induction on Σ^*

Proof cont'd:

Inductive case ($w = aw'$): Assume that $P(w')$ holds. That is, for all $v \in \Sigma^*$:

$$\text{(IH)}: \qquad \text{length}(w'v) = \text{length}(w') + \text{length}(v).$$

Then, for all $a \in \Sigma$, we have:

$$\begin{aligned} \text{length}((aw')v) &= \text{length}(a(w'v)) && \text{(concat.I)} \\ &= 1 + \text{length}(w'v) && \text{(length.I)} \\ &= 1 + \text{length}(w') + \text{length}(v) && \text{(IH)} \\ &= \text{length}(aw') + \text{length}(v) && \text{(length.I)} \end{aligned}$$

So $P(aw')$ holds.

We have $P(\lambda)$ and for all $w' \in \Sigma^*$ and $a \in \Sigma$: $P(w') \to P(aw')$. Hence $P(w)$ holds for all $w \in \Sigma^*$.

Example: Induction on Σ^*

Proof cont'd:

Inductive case ($w = aw'$): Assume that $P(w')$ holds. That is, for all $v \in \Sigma^*$:

$$\text{(IH):} \qquad \text{length}(w'v) = \text{length}(w') + \text{length}(v).$$

Then, for all $a \in \Sigma$, we have:

$$\begin{aligned}
\text{length}((aw')v) &= \text{length}(a(w'v)) && \text{(concat.I)}\\
&= 1 + \text{length}(w'v) && \text{(length.I)}\\
&= 1 + \text{length}(w') + \text{length}(v) && \text{(IH)}\\
&= \text{length}(aw') + \text{length}(v) && \text{(length.I)}
\end{aligned}$$

So $P(aw')$ holds.

We have $P(\lambda)$ and for all $w' \in \Sigma^*$ and $a \in \Sigma$: $P(w') \rightarrow P(aw')$. Hence $P(w)$ holds for all $w \in \Sigma^*$.

Example: Induction on Σ^*

Proof cont'd:

Inductive case ($w = aw'$): Assume that $P(w')$ holds. That is, for all $v \in \Sigma^*$:

$$\text{(IH):} \qquad \text{length}(w'v) = \text{length}(w') + \text{length}(v).$$

Then, for all $a \in \Sigma$, we have:

$$\begin{aligned}
\text{length}((aw')v) &= \text{length}(a(w'v)) && \text{(concat.I)} \\
&= 1 + \text{length}(w'v) && \text{(length.I)} \\
&= 1 + \text{length}(w') + \text{length}(v) && \text{(IH)} \\
&= \text{length}(aw') + \text{length}(v) && \text{(length.I)}
\end{aligned}$$

So $P(aw')$ holds.

We have $P(\lambda)$ and for all $w' \in \Sigma^*$ and $a \in \Sigma$: $P(w') \rightarrow P(aw')$. Hence $P(w)$ holds for all $w \in \Sigma^*$.

Example: Induction on Σ^*

Proof cont'd:

Inductive case ($w = aw'$): Assume that $P(w')$ holds. That is, for all $v \in \Sigma^*$:

$$\text{(IH):} \qquad \text{length}(w'v) = \text{length}(w') + \text{length}(v).$$

Then, for all $a \in \Sigma$, we have:

$$\begin{aligned} \text{length}((aw')v) &= \text{length}(a(w'v)) && \text{(concat.I)} \\ &= 1 + \text{length}(w'v) && \text{(length.I)} \\ &= 1 + \text{length}(w') + \text{length}(v) && \text{(IH)} \\ &= \text{length}(aw') + \text{length}(v) && \text{(length.I)} \end{aligned}$$

So $P(aw')$ holds.

We have $P(\lambda)$ and for all $w' \in \Sigma^*$ and $a \in \Sigma$: $P(w') \rightarrow P(aw')$.
Hence $P(w)$ holds for all $w \in \Sigma^*$.

Example: Induction on Σ^*

Proof cont'd:

Inductive case ($w = aw'$): Assume that $P(w')$ holds. That is, for all $v \in \Sigma^*$:

$$\text{(IH):} \qquad \text{length}(w'v) = \text{length}(w') + \text{length}(v).$$

Then, for all $a \in \Sigma$, we have:

$$\begin{aligned} \text{length}((aw')v) &= \text{length}(a(w'v)) && \text{(concat.I)} \\ &= 1 + \text{length}(w'v) && \text{(length.I)} \\ &= 1 + \text{length}(w') + \text{length}(v) && \text{(IH)} \\ &= \text{length}(aw') + \text{length}(v) && \text{(length.I)} \end{aligned}$$

So $P(aw')$ holds.

We have $P(\lambda)$ and for all $w' \in \Sigma^*$ and $a \in \Sigma$: $P(w') \rightarrow P(aw')$.
Hence $P(w)$ holds for all $w \in \Sigma^*$.

Example: Induction on Σ^*

Proof cont'd:

Inductive case ($w = aw'$): Assume that $P(w')$ holds. That is, for all $v \in \Sigma^*$:

$$\text{(IH):} \qquad \text{length}(w'v) = \text{length}(w') + \text{length}(v).$$

Then, for all $a \in \Sigma$, we have:

$$\begin{aligned} \text{length}((aw')v) &= \text{length}(a(w'v)) && \text{(concat.I)} \\ &= 1 + \text{length}(w'v) && \text{(length.I)} \\ &= 1 + \text{length}(w') + \text{length}(v) && \text{(IH)} \\ &= \text{length}(aw') + \text{length}(v) && \text{(length.I)} \end{aligned}$$

So $P(aw')$ holds.

We have $P(\lambda)$ and for all $w' \in \Sigma^*$ and $a \in \Sigma$: $P(w') \to P(aw')$. Hence $P(w)$ holds for all $w \in \Sigma^*$.

Example 2: Induction on Σ^*

Define $\text{reverse} : \Sigma^* \to \Sigma^*$:

(rev.B) $\text{reverse}(\lambda) = \lambda$,

(rev.I) $\text{reverse}(a \cdot w) = \text{reverse}(w) \cdot a$

Example 2: Induction on Σ^*

Theorem

For all $w, v \in \Sigma^*$, $reverse(wv) = reverse(v) \cdot reverse(w)$.

Proof: By induction on w...

$$\begin{aligned}
\textbf{[B]} \quad \text{reverse}(\lambda v) &= \text{reverse}(v) && \text{(concat.B)}\\
&= \text{reverse}(v)\lambda && (*)\\
&= \text{reverse}(v)\text{reverse}(\lambda) && \text{(reverse.B)}
\end{aligned}$$

$$\begin{aligned}
\textbf{[I]} \quad \text{reverse}((aw')v) &= \text{reverse}(a(w'v)) && \text{(concat.I)}\\
&= \text{reverse}(w'v) \cdot a && \text{(reverse.I)}\\
&= \text{reverse}(v)\text{reverse}(w') \cdot a && \text{(IH)}\\
&= \text{reverse}(v)\text{reverse}(aw') && \text{(reverse.I)}
\end{aligned}$$

Example 2: Induction on Σ^*

Theorem

For all $w, v \in \Sigma^*$, $\mathit{reverse}(wv) = \mathit{reverse}(v) \cdot \mathit{reverse}(w)$.

Proof: By induction on w...

$$\begin{array}{llll}
\textbf{[B]} & \text{reverse}(\lambda v) & = \text{reverse}(v) & (\text{concat.B}) \\
 & & = \text{reverse}(v)\lambda & (*) \\
 & & = \text{reverse}(v)\text{reverse}(\lambda) & (\text{reverse.B}) \\
 & & & \\
\textbf{[I]} & \text{reverse}((aw')v) & = \text{reverse}(a(w'v)) & (\text{concat.I}) \\
 & & = \text{reverse}(w'v) \cdot a & (\text{reverse.I}) \\
 & & = \text{reverse}(v)\text{reverse}(w') \cdot a & (\text{IH}) \\
 & & = \text{reverse}(v)\text{reverse}(aw') & (\text{reverse.I})
\end{array}$$

Example 2: Induction on Σ^*

Theorem

For all $w, v \in \Sigma^*$, $reverse(wv) = reverse(v) \cdot reverse(w)$.

Proof: By induction on w...

[B]	$\text{reverse}(\lambda v)$	$= \text{reverse}(v)$	(concat.B)
		$= \text{reverse}(v)\lambda$	(*)
		$= \text{reverse}(v)\text{reverse}(\lambda)$	(reverse.B)
[I]	$\text{reverse}((aw')v)$	$= \text{reverse}(a(w'v))$	(concat.I)
		$= \text{reverse}(w'v) \cdot a$	(reverse.I)
		$= \text{reverse}(v)\text{reverse}(w') \cdot a$	(IH)
		$= \text{reverse}(v)\text{reverse}(aw')$	(reverse.I)

Mutual Recursion

Several more sophisticated programs employ a technique of two procedures calling each other. Of course, it should be designed so that each consecutive call refers to ever smaller parameters, so that the entire process terminates. This method is often used in computer graphics, in particular for generating fractal images (basis of various imaginary landscapes, among others).

Mutual Recursion

Example

Alternative definition of Fibonacci numbers:

$$\begin{array}{ll} (B) & f(1) = 1 \\ (B) & g(1) = 1 \\ (R) & f(n) = f(n-1) + g(n-1) \\ (R) & g(n) = f(n-1) \end{array}$$

In matrix form:

$$\begin{pmatrix} f(n) \\ g(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f(n-1) \\ g(n-1) \end{pmatrix}$$

Corollary:

$$\begin{pmatrix} f(n) \\ g(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} f(0) \\ g(0) \end{pmatrix}$$

Mutual Recursion

Example

Alternative definition of Fibonacci numbers:

$$\begin{array}{ll} (B) & f(1) = 1 \\ (B) & g(1) = 1 \\ (R) & f(n) = f(n-1) + g(n-1) \\ (R) & g(n) = f(n-1) \end{array}$$

In matrix form:

$$\begin{pmatrix} f(n) \\ g(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f(n-1) \\ g(n-1) \end{pmatrix}$$

Corollary:

$$\begin{pmatrix} f(n) \\ g(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} f(0) \\ g(0) \end{pmatrix}$$

Summary of topics

- Recursion
- Recursive Data Types
- Induction
- Structural Induction

What is assessible?

- Recursive definitions
- Structural induction