COMP2111 Week 3
Term 1, 2024
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

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

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

A recursive definition has base cases (B) and recursive cases (R):

$$
\begin{array}{ll}
(B) & 0! = 1 \\
(R) & (n+1)! = (n+1) \cdot n!
\end{array}
$$

$$
\begin{array}{ll}
 & \texttt{fact}(n)\texttt{:} \\
(B) & \quad \texttt{if}(n = 0)\texttt{: } 1 \\
(R) & \quad \texttt{else: } n * \texttt{fact}(n-1)
\end{array}
$$

Factorial is defined in terms of *smaller* instances of factorial.

Recursion

A recursive definition has base cases (B) and recursive cases (R):

$$
\begin{array}{ll}
(B) & 0! = 1 \\
(R) & (n+1)! = (n+1) \cdot n!
\end{array}
$$

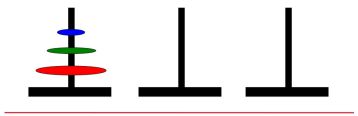
$$
\begin{array}{ll}
 & \texttt{fact}(n)\texttt{:} \\
(B) & \quad \texttt{if}(n = 0)\texttt{: } 1 \\
(R) & \quad \texttt{else: } n * \texttt{fact}(n-1)
\end{array}
$$

Factorial is defined in terms of *smaller* instances of factorial.

Question

Why do we need base cases in programming?
Why do we need them in maths?

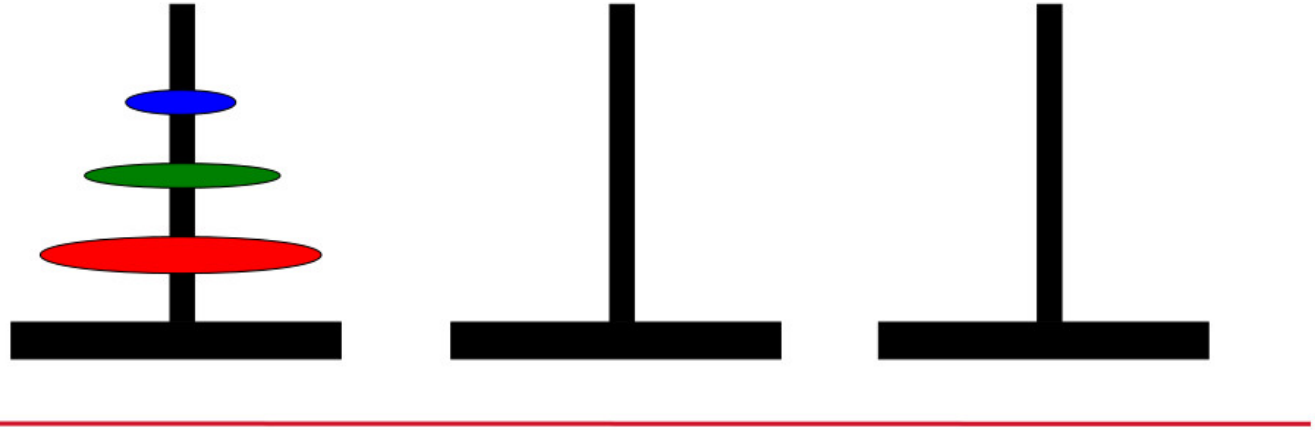
Example: Towers of Hanoi



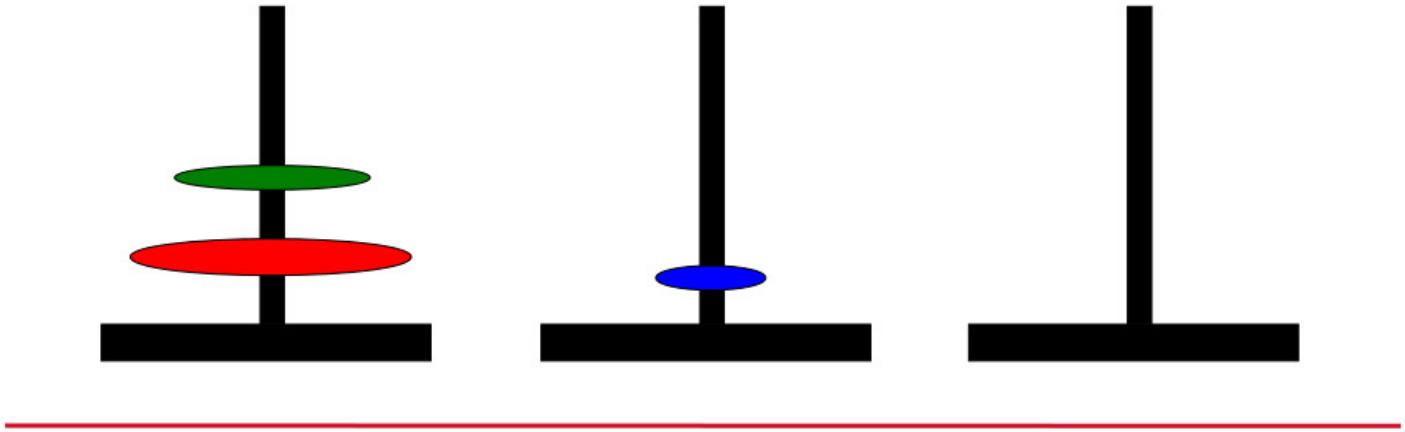
1. There are 3 towers (pegs).
2. n disks of decreasing size are placed on the first tower.
3. Every move, you take the top disk from one peg and put it on top of another peg.
4. You win when all disks are on the middle tower.
5. Larger disks cannot be placed on of smaller disks.

The last tower can be used to temporarily hold disks.

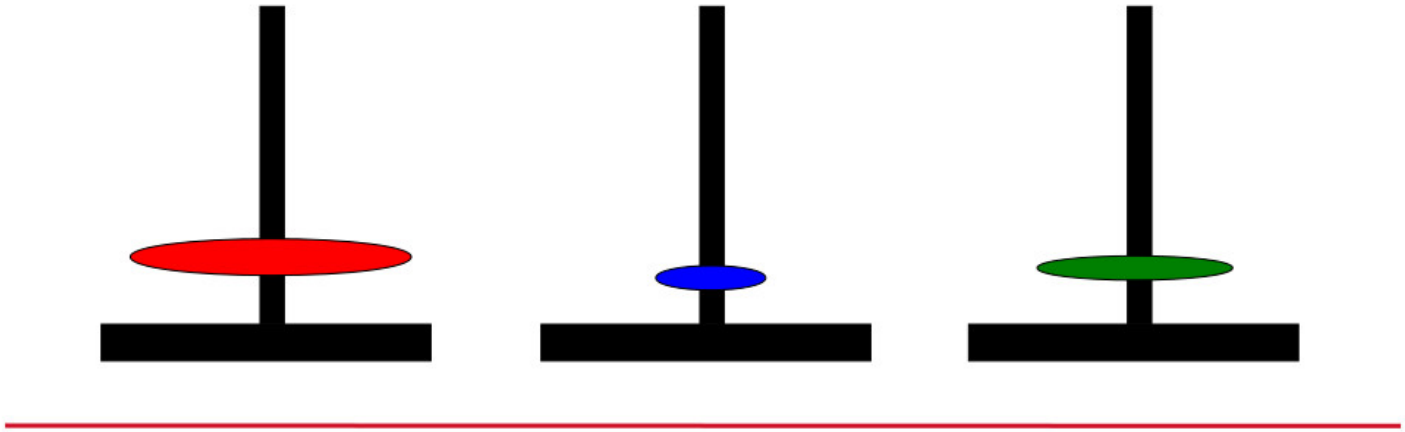
Example: Towers of Hanoi



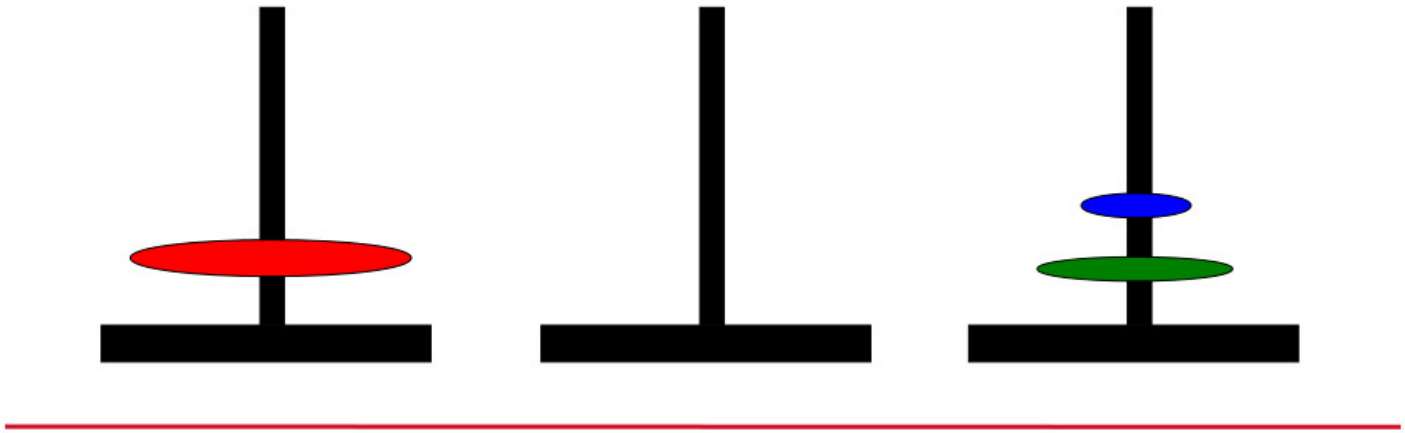
Example: Towers of Hanoi



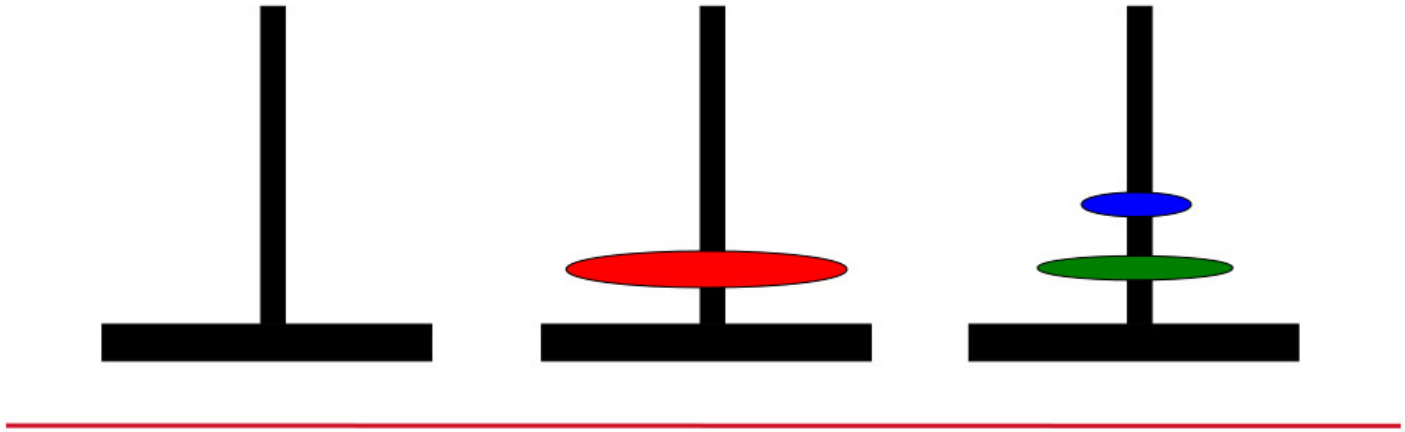
Example: Towers of Hanoi



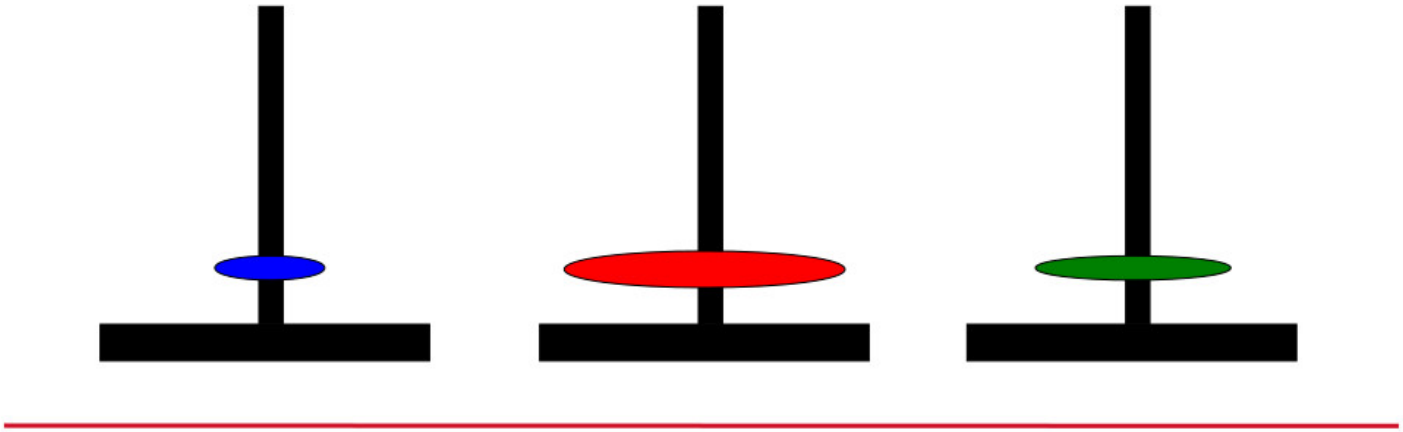
Example: Towers of Hanoi



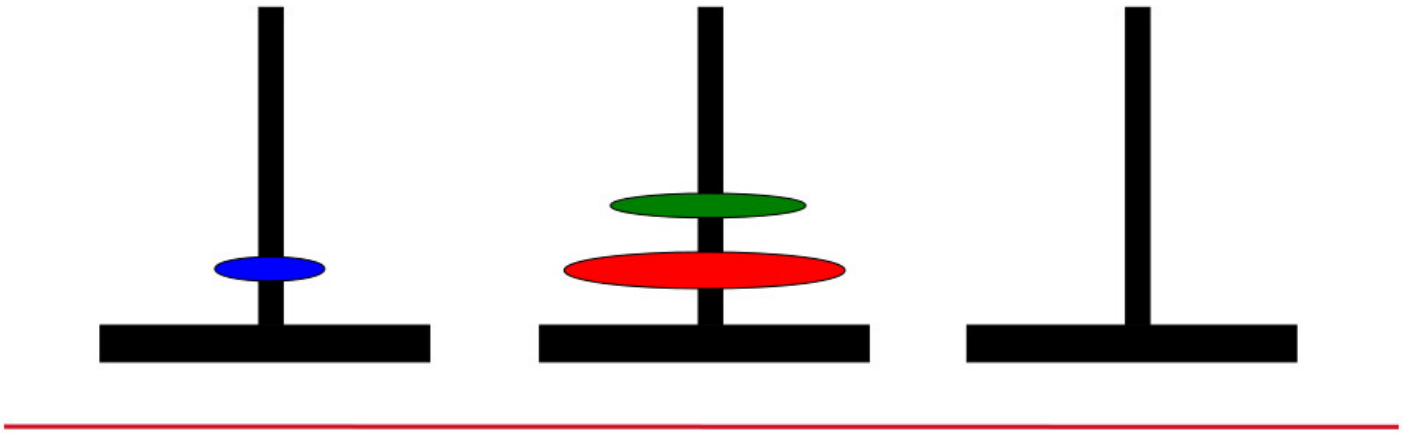
Example: Towers of Hanoi



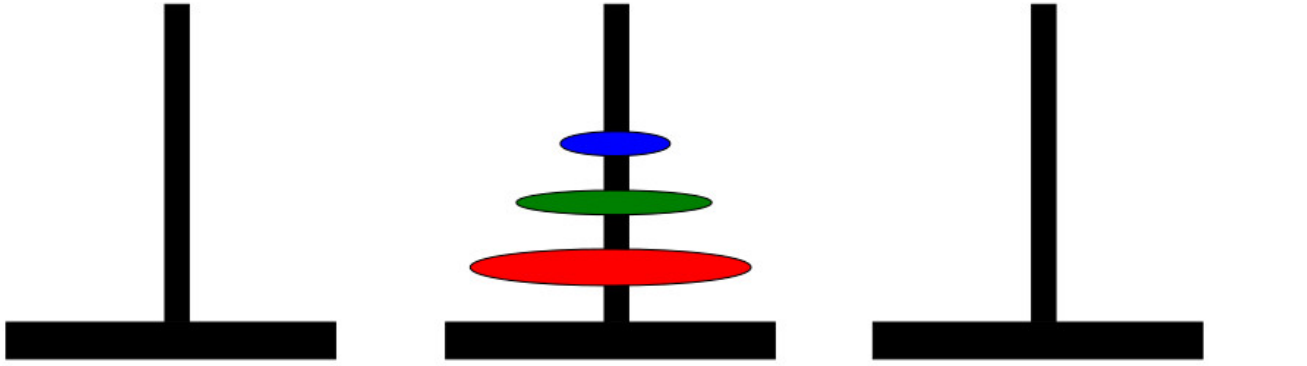
Example: Towers of Hanoi



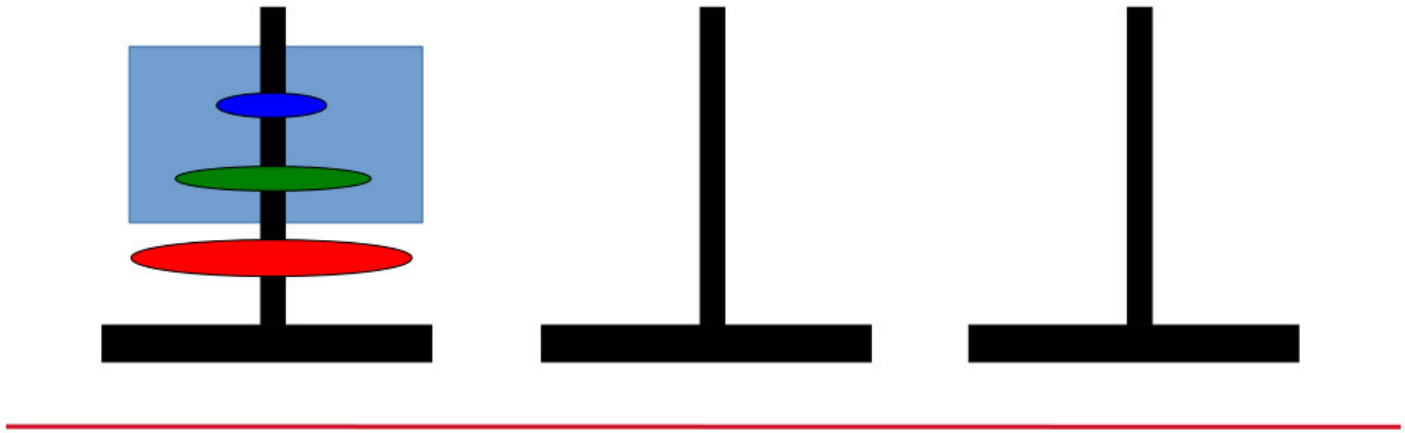
Example: Towers of Hanoi



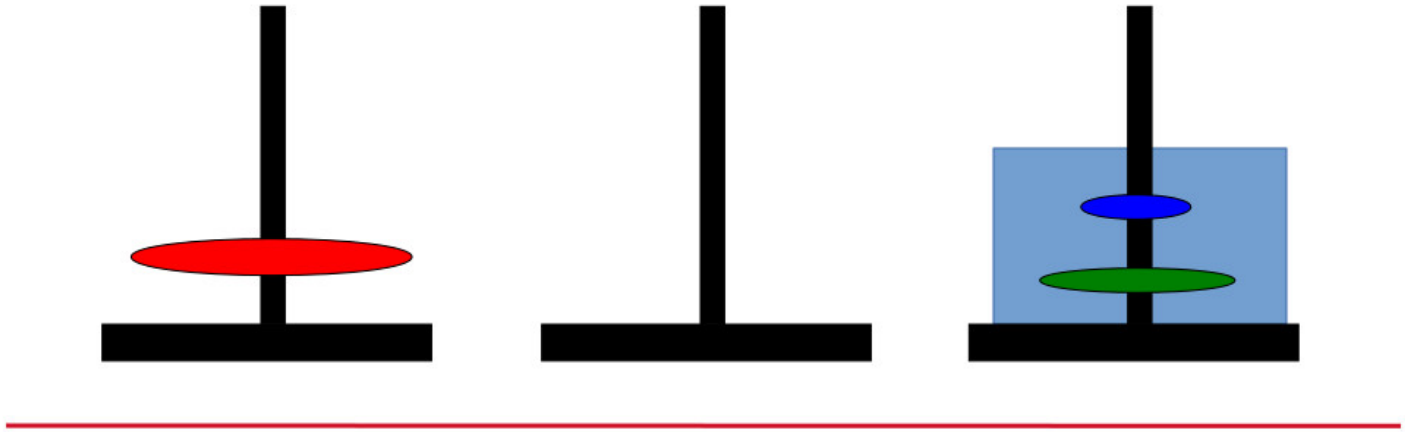
Example: Towers of Hanoi



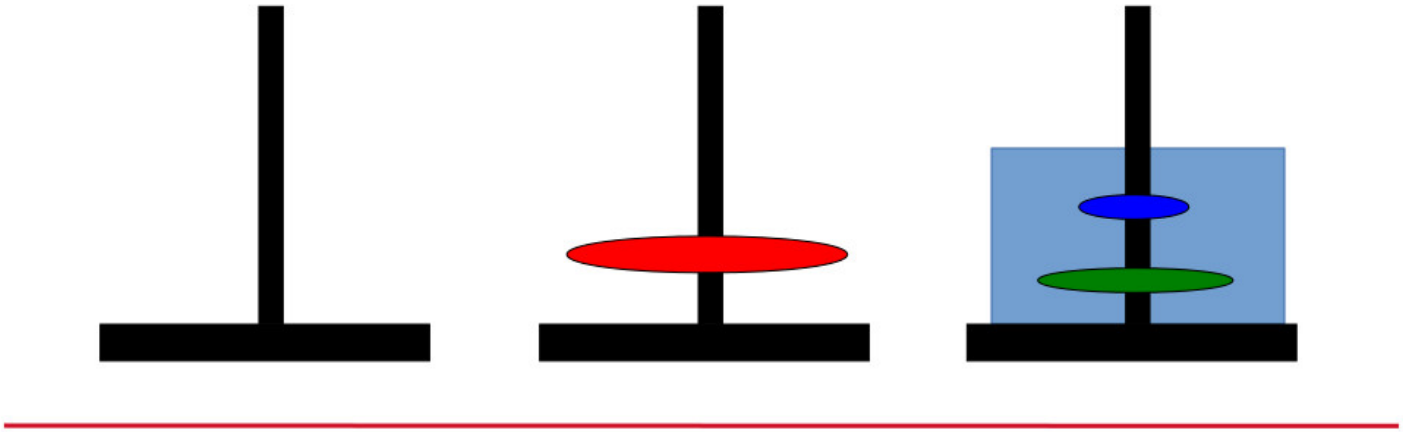
Example: Towers of Hanoi



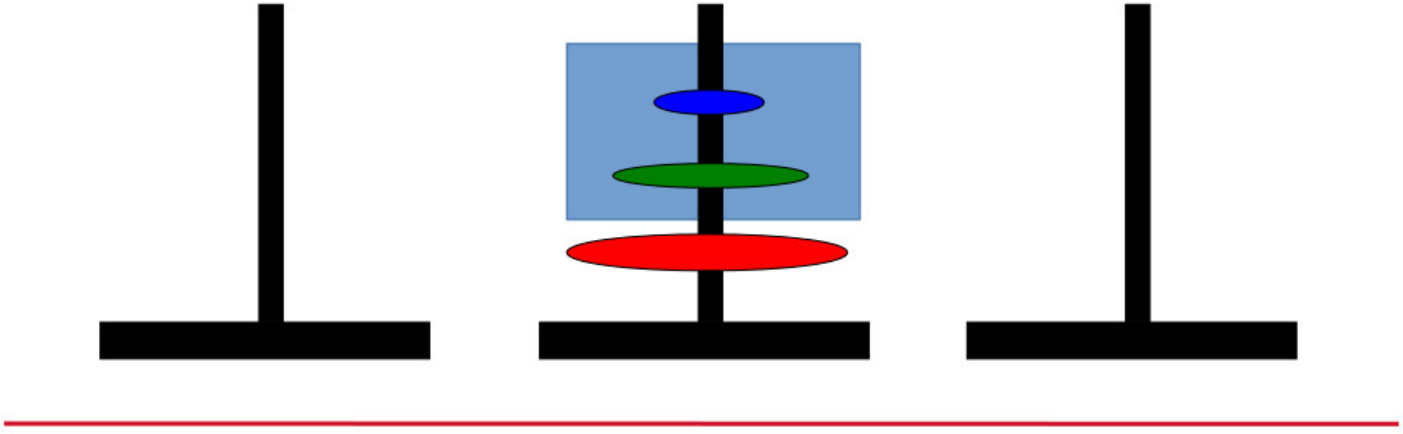
Example: Towers of Hanoi



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).

Formally:

(B) $0 \in \mathbb{N}$

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

This is an **inductive** definition of $\mathbb{N}$ (aka a **recursive** definition): $\mathbb{N}$ contains everything that can be constructed by finitely many applications of (B) and (R), and nothing else.

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) $F_1 = 1$,

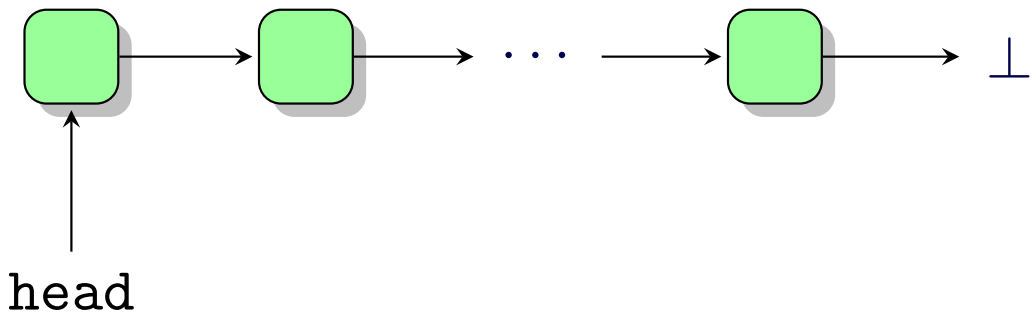
(I) $F_n = F_{n-1} + F_{n-2}$

NB

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

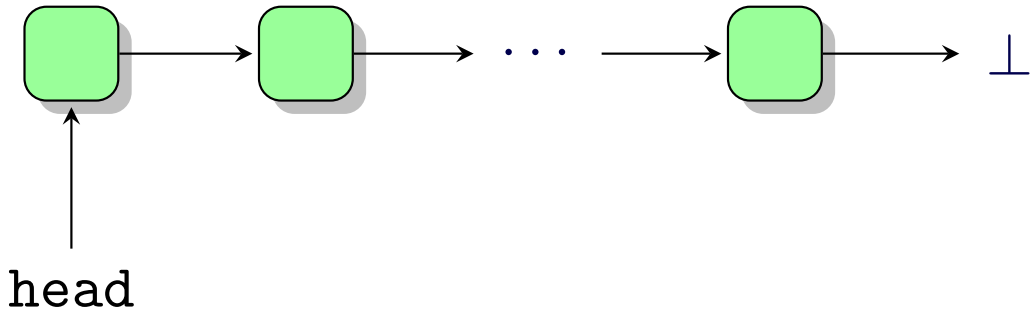
Example: Linked lists

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



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 (`Data`, 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:

$$
\begin{array}{ll}
 & \texttt{fact}(n)\text{:} \\
(B) & \quad \texttt{if}(n = 0)\text{: } 1 \\
(R) & \quad \texttt{else}\text{: } n * \texttt{fact}(n-1)
\end{array}
$$

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

Summing the first n natural numbers:

$$
\begin{array}{ll}
 & \texttt{sum}(n)\text{:} \\
(B) & \quad \texttt{if}(n = 0)\text{: } 0 \\
(R) & \quad \texttt{else}\text{: } n + \texttt{sum}(n-1)
\end{array}
$$

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

Concatenation of words (defining wv):

For all $w, v \in \Sigma^*$ and $a \in \Sigma$:

$$(B) \quad \lambda v = v$$

$$(R) \quad (aw)v = a(wv)$$

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

Length of words:

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

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

$P(x)$ *for all* x (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"

Inductive Reasoning

Suppose we would like to reach a conclusion of the form

$P(x)$ *for all* x (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 where 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: By starting with $a_1 \ldots a_n$ and repeatedly applying (I), we can construct all values in the domain.

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:

$$\begin{aligned}&\texttt{sum}(n):\\ &\quad \texttt{if}(n = 0)\texttt{: } 0\\ &\quad \texttt{else: } n + \texttt{sum}(n-1)\end{aligned}$$

Another attempt:

Example

$$\begin{aligned}&\texttt{sum2}(n):\\ &\quad \texttt{return } n * (n+1)/2\end{aligned}$$

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.

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.

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

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

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

(proof?)

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. Structural induction

Induction From m Upwards

If

(B) $P(m)$

(I) $\forall k \geq m\,(P(k) \rightarrow P(k+1))$

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 $p(m) \to p(m-1)$. 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 from 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.

Summary of topics

- Recursion
- Recursive Data Types
- Induction
- Structural Induction

Structural Induction

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

(B) $0 \in \mathbb{N}$

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

(B) $P(0)$

(I) $P(k) \rightarrow P(k+1)$.

Structural Induction

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

(B) $0 \in \mathbb{N}$

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

(B) $P(0)$

(I) $P(k) \rightarrow P(k+1)$.

NB

Every clause in the induction principle is there because of a similar-looking clause in the (recursive) definition!

The same connection between recursive definition and induction principle applies not just to $\mathbb{N}$, but to any **well-founded strict** poset.

The basic approach is always the same. To prove $\forall x.P(x)$, we show:

(B) P holds for all minimal objects

(I) If P holds for all predecessors of x, then $P(x)$.

Strict poset

A **strict** poset is a pair $(S, \prec)$ consisting of a set S and a relation $\prec \subseteq S \times S$ such that $\prec$ is anti-reflexive, anti-symmetric and transitive.

Example

$(\mathbb{N}, <)$ is a strict poset.

Example

$(\mathbb{N}, \leq)$ is a non-strict poset. (why?)

A (**non-strict**) partial order is reflexive, anti-symmetric and transitive.

Well-founded?

A strict poset $(S, <)$ is **well-founded** if there are no infinitely descending chains:

$$\cdots < r_{k+2} < r_{k+1} < r_k$$

Example

$(\mathbb{N}, <)$ is well-founded: every chain starting from a number n ends in 0 after finitely many steps.

Example

$(\mathbb{Z}, <)$ is **not** well-founded (why?)

Example

$(\mathbb{R}^+, <)$ is **not** well-founded (why?)

Example: Induction on Σ^*

Recall definition of Σ^*:

$\lambda \in \Sigma^*$

If $w \in \Sigma^*$ then $aw \in \Sigma^*$ 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) $\text{length}(aw) = 1 + \text{length}(w)$

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 ($w = \lambda$):

$$\text{length}(\lambda v) \quad =$$

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 ($w = \lambda$):

$$\text{length}(\lambda v) = \text{length}(v) \qquad \text{(concat.B)}$$

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 ($w = \lambda$):

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

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:

$$\text{length}((aw')v) \quad =$$

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:

$$\text{length}((aw')v) \;=\; \text{length}(a(w'v)) \qquad \text{(concat.I)}$$

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:

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

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:

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

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:

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

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.

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

Example 2: Induction on Σ^*

Theorem

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

Proof: By induction on w...

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)

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

(I)

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

Mutual Recursion

Mutual recursion is when two or more functions are defined in terms of each other:

$$
\begin{array}{ll}
 & \texttt{odd}(n)\text{:} \\
(B) & \quad \texttt{if}(n = 0)\text{: } \textit{false} \\
(R) & \quad \texttt{else}\text{: } \texttt{even}(n-1) \\
 & \\
 & \texttt{even}(n)\text{:} \\
(B) & \quad \texttt{if}(n = 0)\text{: } \textit{true} \\
(R) & \quad \texttt{else}\text{: } \texttt{odd}(n-1)
\end{array}
$$

Mutual Recursion

Example

Alternative definition of Fibonacci numbers:

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

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