COMP9020 Lecture 7
Session 2, 2017
Induction and Recursion

Administrivia

- Guidelines for good mathematical writing
- Assignment 1 Solutions now available; marks available soon
- Assignment 2 available on Saturday, **due October 1, 23:59**
- Quiz 7 available tonight

Lecture 6 recap

Logic:

- Boolean algebras
- CNF/DNF
- Karnaugh maps

Lecture 6 recap

Boolean Algebra: $(A, \vee, \wedge, ', 0, 1)$, where

- A is a set
- $\vee$, $\wedge$ are binary operations (functions $A \times A \to A$)
- $'$ is a unary operation (function $A \to A$)
- 0 and 1 are special elements of A (constants)
- Operations and constants satisfy certain properties

Examples:

- Two-valued logic: $(\{\top, \bot\}, \vee, \wedge, \neg, \bot, \top)$
- Subsets of X: $(\mathrm{Pow}(X), \cup, \cap, \cdot^c, \emptyset, X)$

NB

"Boolean algebra" is like a Java interface; examples are like Objects implementing the interface.

Lecture 6 recap

Boolean Algebra: $(A, \vee, \wedge, ', 0, 1)$, where

- A is a set
- $\vee$, $\wedge$ are binary operations (functions $A \times A \to A$)
- $'$ is a unary operation (function $A \to A$)
- 0 and 1 are special elements of A (constants)
- Operations and constants satisfy certain properties

Examples:

- Two-valued logic: $(\{\top, \bot\}, \vee, \wedge, \neg, \bot, \top)$
- Subsets of X: $(\mathrm{Pow}(X), \cup, \cap, \cdot^c, \emptyset, X)$

NB

"Boolean algebra" is like a Java interface; examples are like Objects implementing the interface.

Lecture 6 recap

Induction:

- Basic induction
- Induction steps > 1
- Strong induction
- Backward induction
- Infinite descent
- Forward-backward induction
- Structural induction

Basic induction

Example

Prove that for all $n \in \mathbb{N}$ if X has n elements then $\mathrm{Pow}(X)$ has 2^n elements.

Infinite descent

Example

The square root of 2 is irrational.

Proof.

Suppose that $\sqrt{2} = p/q$ for some p and q. **Assume p, q small as possible [in, e.g. product order])**
Then $p^2 = 2q^2$.
Since p^2 is even p must be even: $p = 2r$.
Therefore $p^2 = (2r)^2 = 4r^2 = 2q^2$.
Therefore $q^2 = 2r^2$, so q must be even: $q = 2s$.
So, from p/q we constructed an equivalent solution $r/s = p/q$ with $r < p, s < q \to$ infinite descent. □

NB

The same idea can be used to show that, for any integer, if the square root is not also an integer then it must be irrational.

Recursion

Fundamental concept in Computer Science

- Recursion in algorithms: Solving problems by reducing to smaller cases
  - Factorial
  - gcd computation
  - Towers of Hanoi
  - Mergesort
- Recursion in data structures: Finite definitions of **arbitrarily large** objects
  - Natural numbers
  - Words
  - Formulas
  - (Rooted) trees, especially 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
    - gcd computation
    - Towers of Hanoi
    - Mergesort
- Recursion in data structures: Finite definitions of **arbitrarily large** objects
    - Natural numbers
    - Words
    - Formulas
    - (Rooted) trees, especially 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
  - gcd computation
  - Towers of Hanoi
  - Mergesort
- Recursion in data structures: Finite definitions of **arbitrarily large** objects
  - Natural numbers
  - Words
  - Formulas
  - (Rooted) trees, especially 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)**

Recursion in programs/algorithms

Example: Factorial

Factorial:

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

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

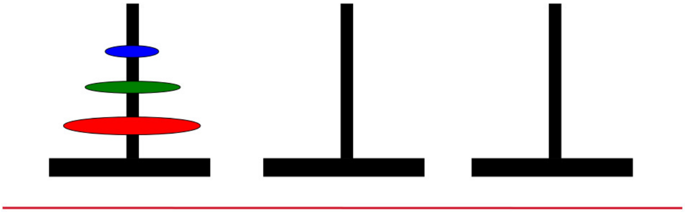
Example: Euclid's gcd algorithm

$$gcd(m, n) = \begin{cases} m & \text{if } m = n \\ gcd(m - n, n) & \text{if } m > n \\ gcd(m, n - m) & \text{if } m < n \end{cases}$$

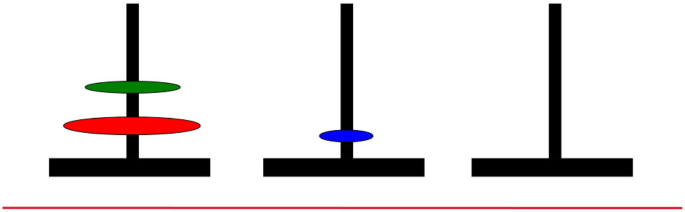
Example: Towers of Hanoi

- 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

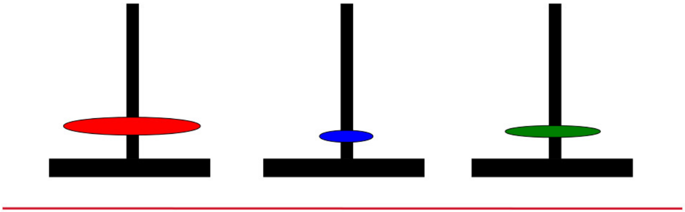
Example: Towers of Hanoi



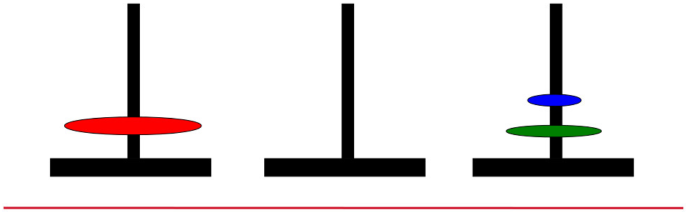
Example: Towers of Hanoi



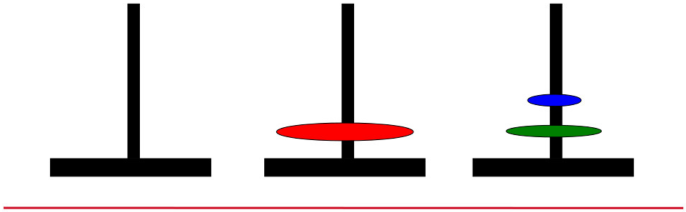
Example: Towers of Hanoi



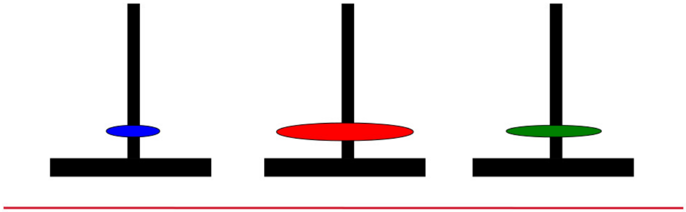
Example: Towers of Hanoi



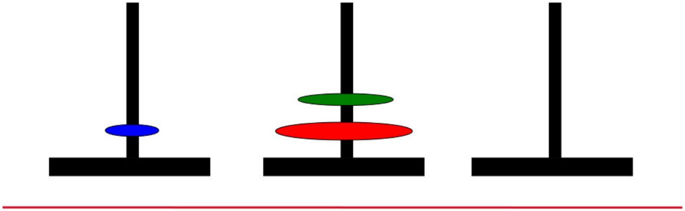
Example: Towers of Hanoi



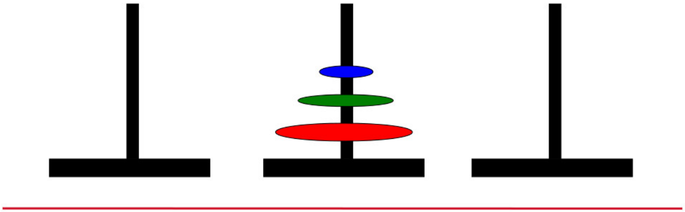
Example: Towers of Hanoi



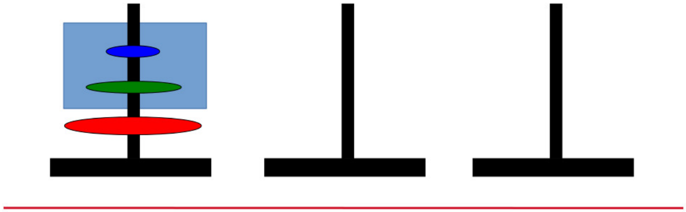
Example: Towers of Hanoi



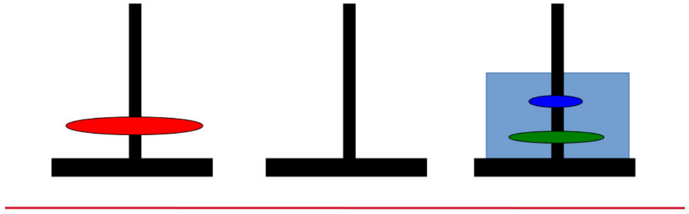
Example: Towers of Hanoi



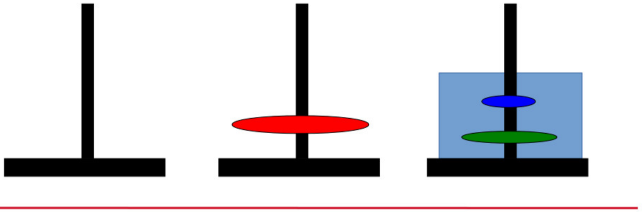
Example: Towers of Hanoi



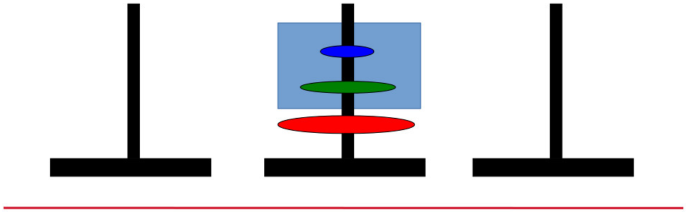
Example: Towers of Hanoi



Example: Towers of Hanoi



Example: Towers of Hanoi



Example: Mergesort

- A list of 1 element is sorted
- A list of n elements can be sorted by recursively sorting each half (two lists of $n/2$ elements) and *merging* the lists together.

```
        mergesort($A$):
$(B)$     if($|A| = 1$): return $A$
          else:
$(R)$       $L$ = mergesort($A[1..n/2]$)
$(R)$       $R$ = mergesort($A[n/2..n]$)
            return merge($L, R$)
```

Recursive data types

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: 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 $a \cdot w \in \Sigma^*$ for all $a \in \Sigma$

NB

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

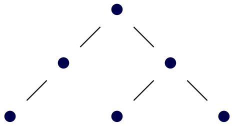
Example: Propositional formulas

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.

Example: Rooted trees

A binary (rooted) tree is either empty (B), or it is a node with two subtrees as children (R). (Note: a child may be empty!)



A rooted tree is either a leaf – i.e. has no children (B), or it has several (rooted) subtrees as children (R).

$$T = \langle r;\ T_1, T_2, \ldots, T_k \rangle$$

Functions over recursive datatypes

Recursive definitions of datatypes naturally lead to recursively defined functions (or programs):

Example

Factorial function `fact` : $\mathbb{N} \to \mathbb{N}$

length : $\Sigma^* \to \mathbb{N}$ can be formally defined as:

- $\text{length}(\lambda) = 0$
- $\text{length}(a \cdot w) = 1 + \text{length}(w)$

A program/function for counting the leaves of a rooted tree:

- $\text{leaves}(\langle r \rangle) = 1$
- $\text{leaves}(\langle r;\ T_1, T_2, \ldots, T_k \rangle) = \text{leaves}(T_1) + \text{leaves}(T_2) + \ldots + \text{leaves}(T_k)$

Functions over recursive datatypes

Example

"Evaluation" of a propositional formula

A program/function for sorting (insertion sort): $\mathsf{sort} : \Sigma^* \to \Sigma^*$:

- $\mathsf{sort}(\lambda) = \lambda$
- $\mathsf{sort}(a \cdot w) = \mathsf{insert}(a, \mathsf{sort}(w))$

Another example (quicksort): $\mathsf{sort} : \Sigma^* \to \Sigma^*$:

- $\mathsf{sort}(\lambda) = \lambda$
- $\mathsf{sort}(a \cdot w) = \mathsf{sort}(w^{<}) \cdot a \cdot \mathsf{sort}(w^{>})$ where $w^{<}$ are the symbols of w less than a and $w^{>}$ is the symbols of w greater than a.

Analysing recursive definitions

Recursive sequences

Asking questions like "how many moves?" leads to recursively defined functions from $\mathbb{N}$ to $\mathbb{N}$. These are also known as *recursive sequences*.

Example

Fibonacci numbers:

$$\begin{array}{ll} (B) & \textsc{fib}(1) = 1 \\ (B) & \textsc{fib}(2) = 1 \\ (R) & \textsc{fib}(n) = \textsc{fib}(n-1) + \textsc{fib}(n-2) \end{array}$$

Example: Towers of Hanoi

How many moves to move n disks, $M(n)$?
$M(1) = 1$, $M(2) = 3$, $M(n) = ?$

Using the recursive solution:

$$M(n) = M(n-1) + 1 + M(n-1) = 2.M(n-1) + 1$$

Formula for $M(n)$?

$$M(n) = 2^{n+1} - 1$$

Proof?

Example: Towers of Hanoi

How many moves to move n disks, $M(n)$?
$M(1) = 1$, $M(2) = 3$, $M(n) =?$

Using the recursive solution:

$$M(n) = M(n-1) + 1 + M(n-1) = 2.M(n-1) + 1$$

Formula for $M(n)$?

$$M(n) = 2^{n+1} - 1$$

Proof?

Example: Towers of Hanoi

How many moves to move n disks, $M(n)$?
$M(1) = 1$, $M(2) = 3$, $M(n) = ?$

Using the recursive solution:

$$M(n) = M(n-1) + 1 + M(n-1) = 2.M(n-1) + 1$$

Formula for $M(n)$?

$$M(n) = 2^{n+1} - 1$$

Proof?

Inductive Proofs About Recursive Definitions

Proofs about recursively defined function very often proceed by a mathematical induction following the structure of the definition.

Example

$\forall n \in \mathbb{N} \left(n! \geq 2^{n-1}\right)$

Proof.

[B] $0! = 1 \geq \frac{1}{2} = 2^{0-1}$

[I] Assume $n \geq 1$.

$$\begin{aligned} (n+1)! = n! \cdot (n+1) &\geq 2^{n-1} \cdot (n+1) && \text{by Ind. Hyp.} \\ &\geq 2^{n-1} \cdot 2 && \text{by } n \geq 1 \\ &= 2^n \end{aligned}$$

$\square$

Exercise

$\boxed{4.4.2}$ Define $s_1 = 1$ and $s_{n+1} = \frac{1}{1+s_n}$ for $n \geq 1$
Then $s_1 = 1,\ s_2 = \frac{1}{2},\ s_3 = \frac{2}{3},\ s_4 = \frac{3}{5},\ s_5 = \frac{5}{8}, \ldots$
The numbers in numerator and denominator remind one of the Fibonacci sequence.
Prove by induction that

$$s_n = \frac{\mathrm{FIB}(n)}{\mathrm{FIB}(n+1)}$$

Example (continued)

Furthermore,

$$\lim_{n\to\infty} s_n = \frac{2}{\sqrt{5}+1} = \frac{\sqrt{5}-1}{2} \approx 0.6$$

This is obtained by showing (using induction!) that

$$\mathrm{FIB}(n) = \frac{1}{\sqrt{5}}(r_1^n - r_2^n)$$

where $r_1 = \frac{1+\sqrt{5}}{2}$ and $r_2 = \frac{1-\sqrt{5}}{2}$

Exercise

$\boxed{4.4.4}$ (a) Give a recursive definition for the sequence

$$(2,\ 4,\ 16,\ 256,\ \dots)$$

To generate $a_n = 2^{2^n}$ use $a_n = (a_{n-1})^2$.
(The related "Fermat numbers" $F_n = 2^{2^n} + 1$ are used in cryptography.)

(b) Give a recursive definition for the sequence

$$(2,\ 4,\ 16,\ 65536,\ \dots)$$

To generate a "stack" of n 2's use $b_n = 2^{b_{n-1}}$.
(These are *Ackermann's numbers*, first used in logic. The inverse function is extremely slow growing; it is important for the analysis of several data organisation algorithms.)

Exercise

$\boxed{4.4.4}$ (a) Give a recursive definition for the sequence

$$(2,\ 4,\ 16,\ 256,\ \ldots)$$

To generate $a_n = 2^{2^n}$ use $a_n = (a_{n-1})^2$.
(The related "Fermat numbers" $F_n = 2^{2^n} + 1$ are used in cryptography.)

(b) Give a recursive definition for the sequence

$$(2,\ 4,\ 16,\ 65536,\ \ldots)$$

To generate a "stack" of n 2's use $b_n = 2^{b_{n-1}}$.
(These are *Ackermann's numbers*, first used in logic. The inverse function is extremely slow growing; it is important for the analysis of several data organisation algorithms.)

Correctness of Recursive Definition

A recurrence formula is correct if the computation of any later term can be reduced to the initial values given in (B).

Example (Incorrect definition)

- Function $g(n)$ is defined recursively by

$$g(n) = g(g(n-1)-1)+1, \qquad g(0) = 2.$$

The definition of $g(n)$ is incomplete — the recursion may not terminate:
Attempt to compute $g(1)$ gives

$$g(1) = g(g(0)-1)+1 = g(1)+1 = \ldots = g(1)+1+1+1\ldots$$

When implemented, it leads to an overflow; most static analyses cannot detect this kind of ill-defined recursion.

Example (continued)

However, the definition could be repaired. For example, we can add the specification specify $g(1) = 2$.

Then $g(2) = g(2-1) + 1 = 3$,
$g(3) = g(g(2) - 1) + 1 = g(3-1) + 1 = 4$,
$\ldots$

In fact, by induction ... $g(n) = n + 1$

This illustrates a very important principle: the boundary (limiting) cases of the definition are evaluated *before* applying the recursive construction.

Example

Function $f(n)$ is defined by

$$f(n) = f(\lceil n/2 \rceil), \quad f(0) = 1$$

When evaluated for $n = 1$ it leads to

$$f(1) = f(1) = f(1) = \dots$$

This one can also be repaired. For example, one could specify that $f(1) = 1$.
This would lead to a constant function $f(n) = 1$ for all $n \geq 0$.

Structural Induction

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

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

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

Example: Induction on Rooted Trees

We write $T = \langle r;\ T_1, T_2, \ldots, T_k \rangle$ for a tree T with root r and k subtrees at the root $T_1, \ldots, T_k$

If

[B]	$p(\langle v; \rangle)$	for trees with only a root
[I]	$p(T_1) \wedge \ldots \wedge p(T_k) \rightarrow p(T)$	for all trees $T = \langle r;\ T_1, T_2, \ldots, T_k \rangle$

then

[C] $p(T)$ for every tree T

Example

Theorem

In any rooted tree the number of vertices is one more than the number of edges.

Proof.

[B] If $T = \langle v; \rangle$ then $v(T) = 1$ and $e(T) = 0$

$\square$

Example

Theorem

In any rooted tree the number of vertices is one more than the number of edges.

Proof.

[B] If $T = \langle v; \rangle$ then $v(T) = 1$ and $e(T) = 0$

[I] If $T = \langle r;\ T_1, T_2, \ldots, T_k \rangle$ then
$v(T) = 1 + \sum_{i=1}^{k} v(T_i)$ and $e(T) = k + \sum_{i=1}^{k} e(T_i)$

$\square$

Example

Theorem

In any rooted tree the number of vertices is one more than the number of edges.

Proof.

[B] If $T = \langle v; \rangle$ then $v(T) = 1$ and $e(T) = 0$

[I] If $T = \langle r;\ T_1, T_2, \ldots, T_k \rangle$ then

$$v(T) = 1 + \textstyle\sum_{i=1}^{k} v(T_i) \quad \text{and} \quad e(T) = k + \sum_{i=1}^{k} e(T_i)$$

From the Ind. Hyp. on $T_1, \ldots, T_k$ it follows that

$$\textstyle\sum_{i=1}^{k} v(T_i) = \sum_{i=1}^{k}(e(T_i) + 1) = (\sum_{i=1}^{k} e(T_i)) + k$$

Therefore

$$v(T) = 1 + (\textstyle\sum_{i=1}^{k} e(T_i)) + k = 1 + e(T)$$

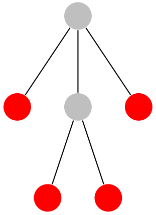
□

Example

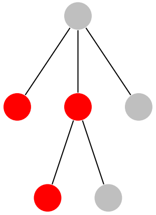
Theorem

In any rooted tree the number of leaves is one more than the number of vertices that have a right sibling.

Proof: exercise



4 leaves



3 vertices with right sibling

Example: Induction on Σ^*

Formal definition of Σ^*:

- $\lambda \in \Sigma^*$
- If $w \in \Sigma^*$ then $a \cdot w \in \Sigma^*$ for all $a \in \Sigma$

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

- $\text{reverse}(\lambda) = \lambda$,
- $\text{reverse}(a \cdot w) = \text{reverse}(w) \cdot a$

Example: Induction on Σ^*

Formal definition of Σ^*:

- $\lambda \in \Sigma^*$
- If $w \in \Sigma^*$ then $a \cdot w \in \Sigma^*$ for all $a \in \Sigma$

Define reverse : $\Sigma^* \to \Sigma^*$:

- reverse$(\lambda) = \lambda$,
- reverse$(a \cdot w) =$ reverse$(w) \cdot a$

Example: 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...

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

Example: Induction on Σ^*

Theorem

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

Proof: By induction on w...

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

Example: Induction on Σ^*

Theorem

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

Proof: By induction on w...

[B]
$$\begin{aligned} \text{reverse}(\lambda \cdot v) &= \text{reverse}(v) && \text{[Def]} \\ &= \text{reverse}(v) \cdot \lambda && \text{[Def)]} \\ &= \text{reverse}(\lambda) \cdot \text{reverse}(w) && \text{[Def]} \end{aligned}$$

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

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

- Mathematical induction:
  base case(s), inductive hypothesis $P(k)$,
  inductive step $\forall k\,(P(k) \rightarrow P(k+1))$, conclusion
- Variations:
  strong ind., forward-backward ind., ind. by cases,
  structural ind.
- Recursive definitions