CSI 2101- Mathematical Induction

Many statements assert that a property of the form P(n) is true for all integers n.

Examples:

For every positive integer n: $n! \leq n^n$

Every set with n elements, has 2^n Subsets.

Induction is one of the most important techniques for proving statements of that form.

Mathematical Induction

For example, consider the following algorithm:

```
sum = 0;
for(i=1; i≤n; i++) {
     sum = sum + 2*i-1;
}
```

What is its output?

- n=1 … 1
- n=2 … 1+3 = 4
- n=3 … 1+3+5 = 9
- n=4 … 1+3+5+7 = 16

We suspect that the output is **n^2**

- but how to prove it?

Mathematical Induction

Use induction to prove that the sum of the first n odd integers is n^2.

What's the hypothesis? $P(n)$ – sum of first n odd integers = n^2.

Base case (n=1): the sum of the first 1 odd integer is 1^2.

Since $1 = 1^2$ ☺

Prove a base case (n=1)

Prove $P(k) \rightarrow P(k+1)$

Inductive Step: show that $\forall(k)\ P(k) \rightarrow P(k+1)$.

How? Assume P(k): the sum of the first k odd integers is k^2.

That is assume that $1 + 3 + \ldots + (2k - 1) = k^2$

And prove P(k+1): the sum of the first (k+1) odd integers is $(k+1)^2$.

$$\underbrace{1 + 3 + \ldots + (2k-1)}_{= k^2} + (2k+1) = k^2 + (2k + 1) = (k+1)^2.$$

By inductive hypothesis

Therefore P(k+1) is true

QED

Mathematical Induction

What did we do?

- basic step:
  - prove $P(1)$
- inductive step:
  - assume $P(n)$ and prove $P(n+1)$ (i.e prove $P(n) \rightarrow P(n+1)$)

- Mathematical Induction is a rule of inference that tells us:

- $P(1)$
- $\forall k\ (P(k) \rightarrow P(k+1))$
- --------------------------
- $\therefore\ \forall n\ P(n)$

Why Mathematical Induction works?

It is enough to prove that this rule of inference is valid

Definition: A set S is "well-ordered" if every non-empty subset of S has a least element.

Given (we take as an axiom):

the set of natural numbers (N) is well-ordered.

- Is the set of integers (Z) well ordered?
- Is the set of non-negative reals (R) well ordered?

No.
$\{ x \in R : x > 1 \}$ has no least element.

No.
$\{ x \in Z : x < 0 \}$ has no least element.

Proof that Mathematical Induction Works

By contradiction using the Well-Ordered-Principle. Assume that Mathematical Induction does not work.

We assume that both hypothesis, i.e. the basic step **$P(1)$** and the induction step (**$P(k) \rightarrow P(k+1)$**) are both true but there still exists **a** such that $\neg$ **$P(a)$.**

Let **S** be the set of all elements **x** for which **$\neg P(x)$.**

By the well ordered principle, **S** has a smallest element **a.**

Because **$P(1)$**, we know that **$a \neq 1$.** Therefore we can consider **$b = a-1$.**

Because **a** was the smallest element of **S, b** is not in **S**. Therefore **$P(b)$** holds. By modus ponens using the induction step, we get **$P(a)$**, which is a contradiction

Writing a Proof by Induction

State the hypothesis very clearly:

P(n) is true for all integers $n \geq b$ – state the property P in English

Identify the base case

P(b) holds because …

Inductive Step - Assuming the inductive hypothesis P(k), prove that P(k+1) holds; i.e., $P(k) \rightarrow P(k+1)$

Conclusion: By induction we have shown that P(k) holds for all k>b (b is what was used for the base case).

Mathematical Induction

Another example: Use induction to prove that the

$1 + 2 + 2^2 + \ldots + 2^n = 2^{n+1} - 1$, for all non-negative integers n.

$\forall n \geq 0$ P(n) is true, where P(n): $1 + 2 + 2^2 + \ldots + 2^n = 2^{n+1} - 1$

1 - Base case $n = 0 \quad 1^0 = 2^1 - 1.$ Prove P(0)

Prove $P(k) \rightarrow P(k+1)$

2 – Inductive Hypothesis

Assume P(k) that is, $1 + 2 + 2^2 + \ldots + 2^k = 2^{k+1} - 1$ and prove P(k+1).

$1 + 2 + 2^2 + \ldots + 2^k + 2^{k+1} = 2^{k+1} + 2^{k+1} - 1 = 2*2^{k+1} - 1 = 2^{k+2} - 1$

$= 2^{k+1} - 1$ By inductive hypothesis

Therefore P(k+1) is true

Mathematical Induction

Another example:

Prove P(n) using induction where

P(n): a set S with n elements has 2^n subsets.

Example: if S={1, 2, 3} then S has $8=2^3$ subsets, these are

{∅}, {1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}

Proof by induction that P(n) is true for all $n \geq 0$

1- Base case P(0): a set S with 0 elements has $2^0=1$ subsets.

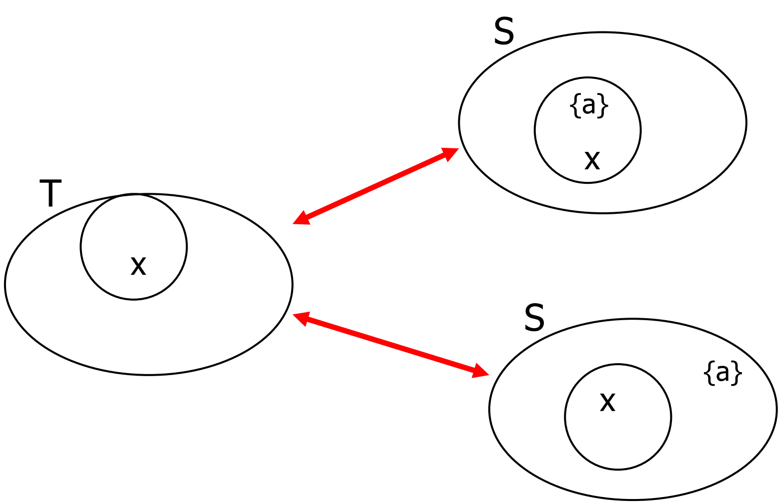
S={ø}, then S has a unique subset {ø} and thus P(0) is true

Mathematical Induction

2- Inductive Step: $\forall(k)\ P(k) \rightarrow P(k+1)$, i.e, assuming P(k) is true we must show that P(k+1) is true.

Assume that any set with k elements has 2^k subsets. Let S a set with k+1 elements. Thus $S = T \cup \{a\}$, where T is a set with k elements.

For each subset X of T there are exactly two subsets of S, namely X and $X \cup \{a\}$. Because there are 2^k subsets of S (inductive hypothesis), there are $2 \times 2^k = 2^{k+1}$ subsets of T.



Therefore P(k+1) is true

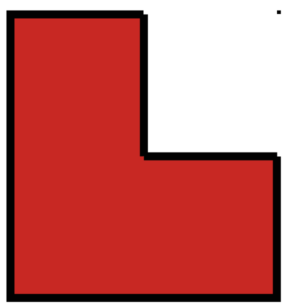
QED

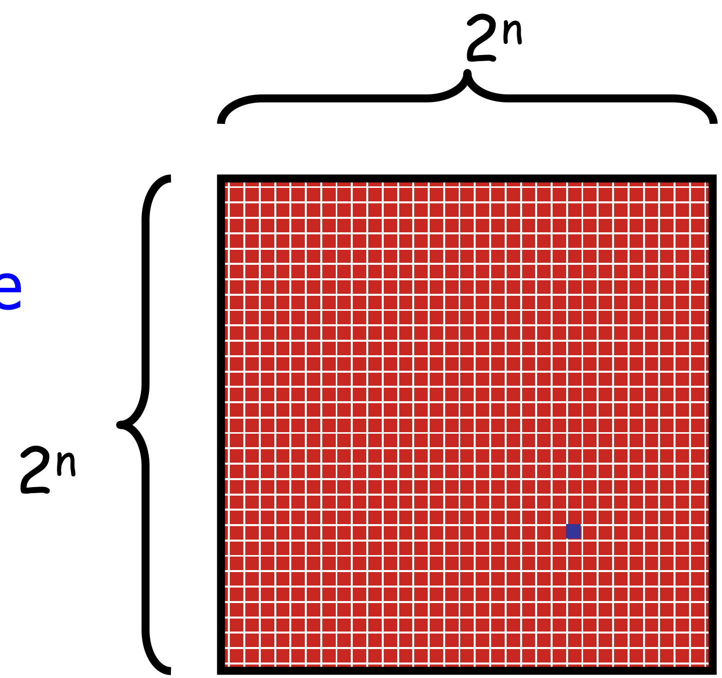
Generating subsets of a set S with k+1 elements from a set T with k elements

Mathematical Induction: Deficient Tiling

A 2^n x 2^n sized grid is *deficient* if all but one cell is tiled.

P(n): all 2n x 2n sized deficient grids can be tiled with right triominoes, which are pieces that cover three squares at a time, like this:



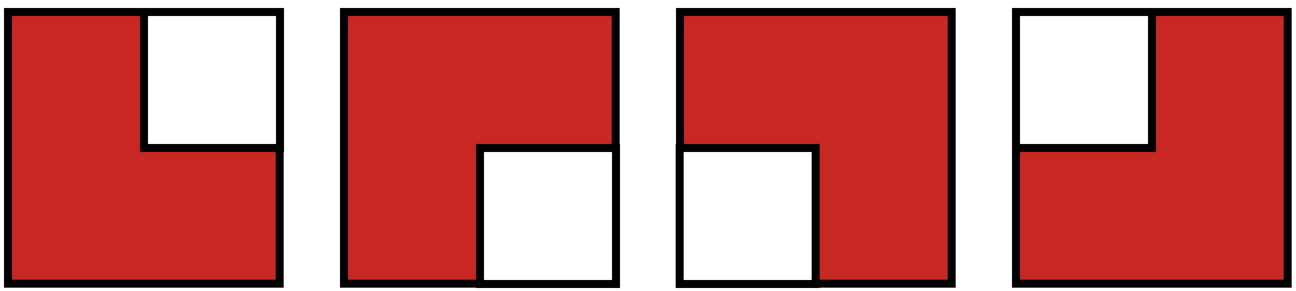


We want to show that for all $n \geq 2$, P(n) is true

Mathematical Induction: Deficient Tiling

- Base Case:

P(1) - Is it true for $2^1 \times 2^1$ grids?

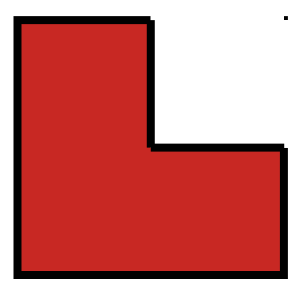


YES ☺

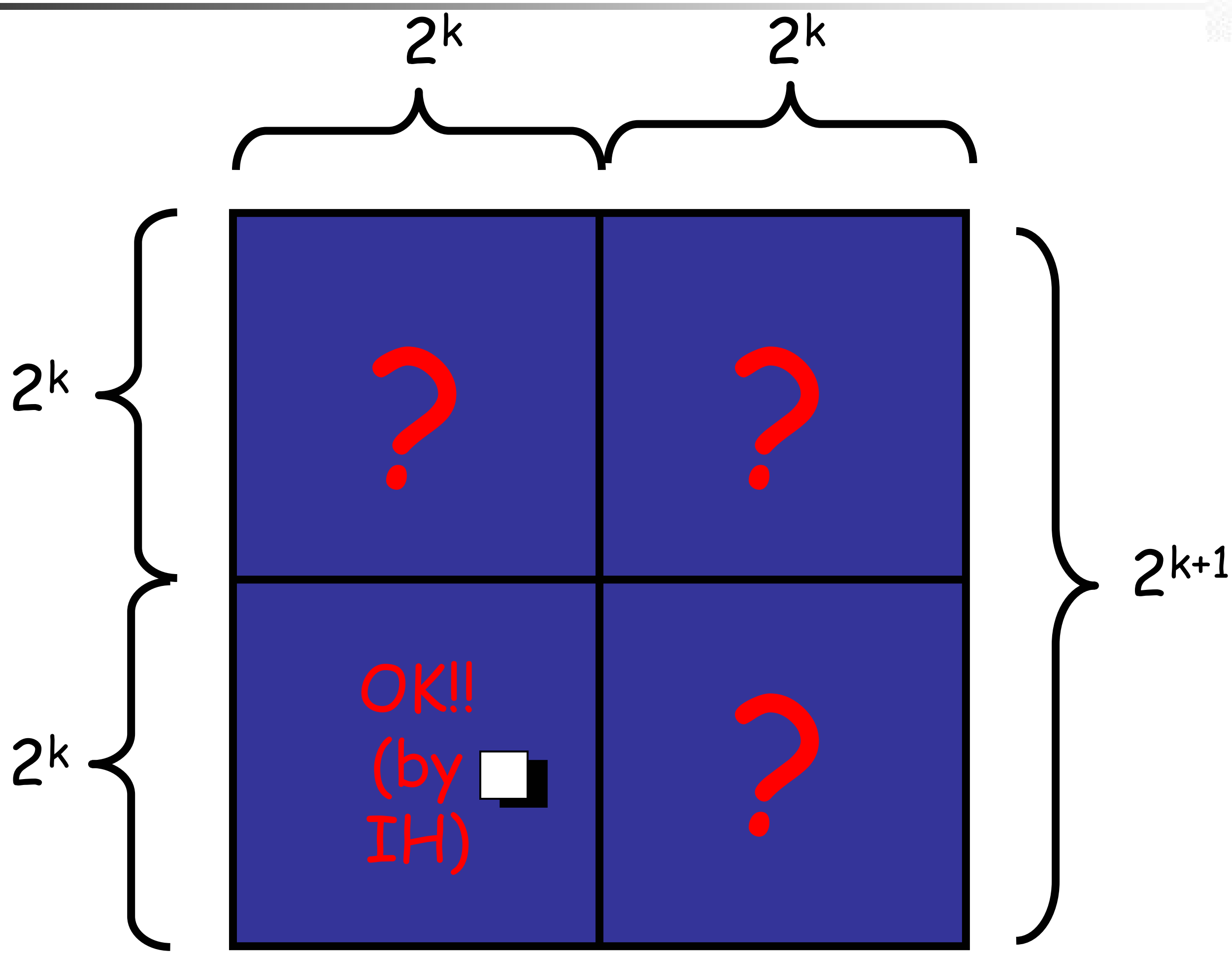
Mathematical Induction: Deficient Tiling

Inductive Step:

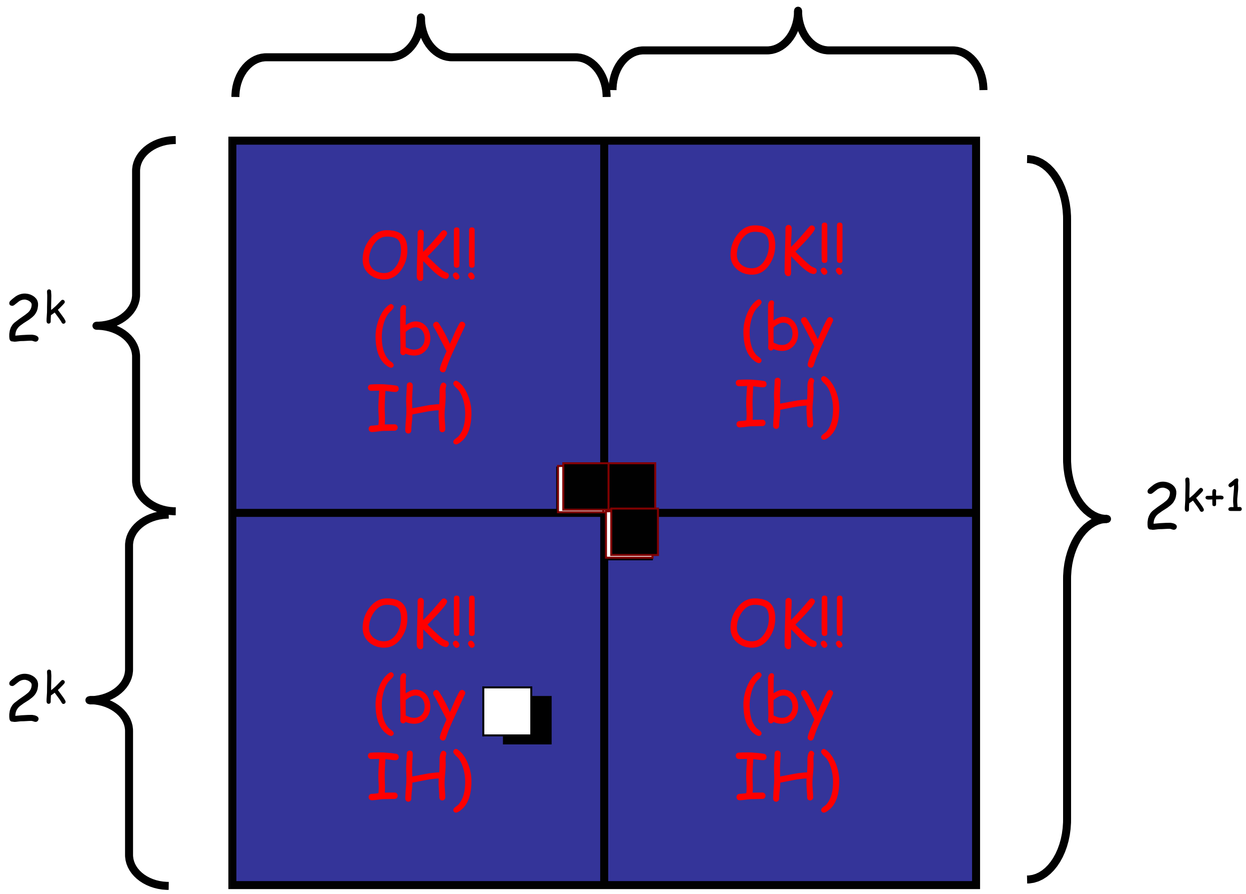
We assume that we can tile a $2^k \times 2^k$ deficient board using our designer tiles. We use this to prove that we can tile a $2^{k+1} \times 2^{k+1}$ deficient board using our designer tiles.



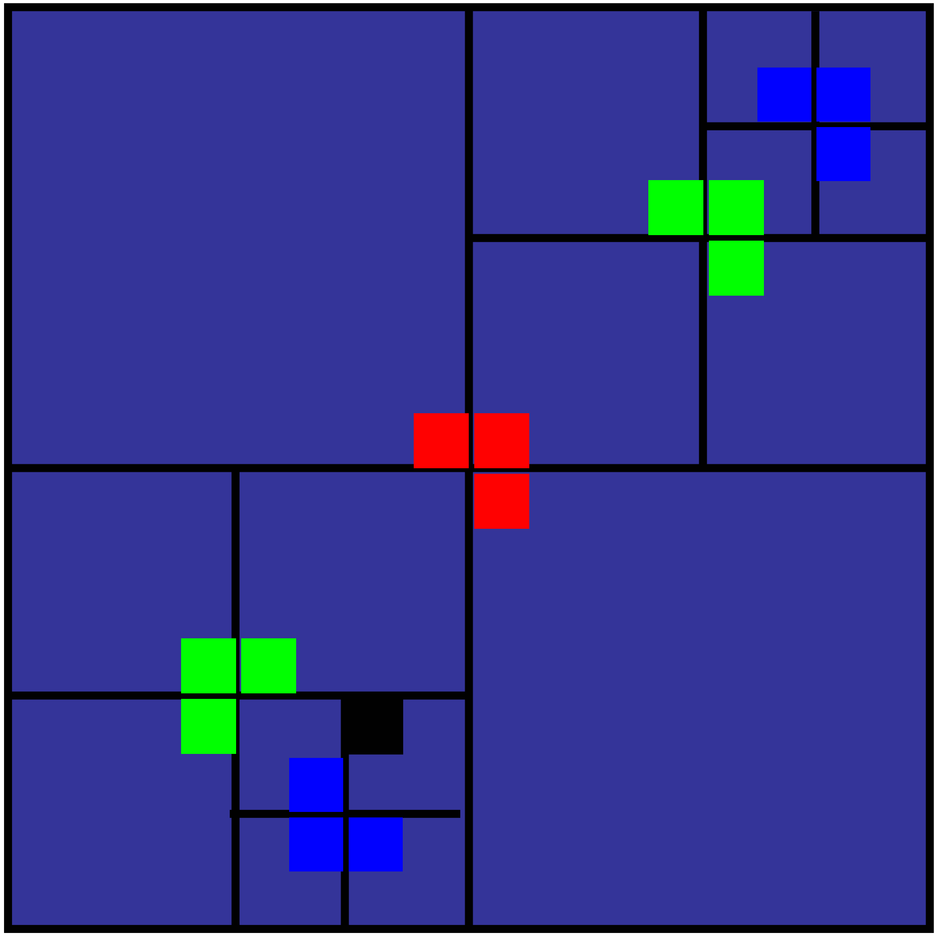
Mathematical Induction: Deficient Tiling



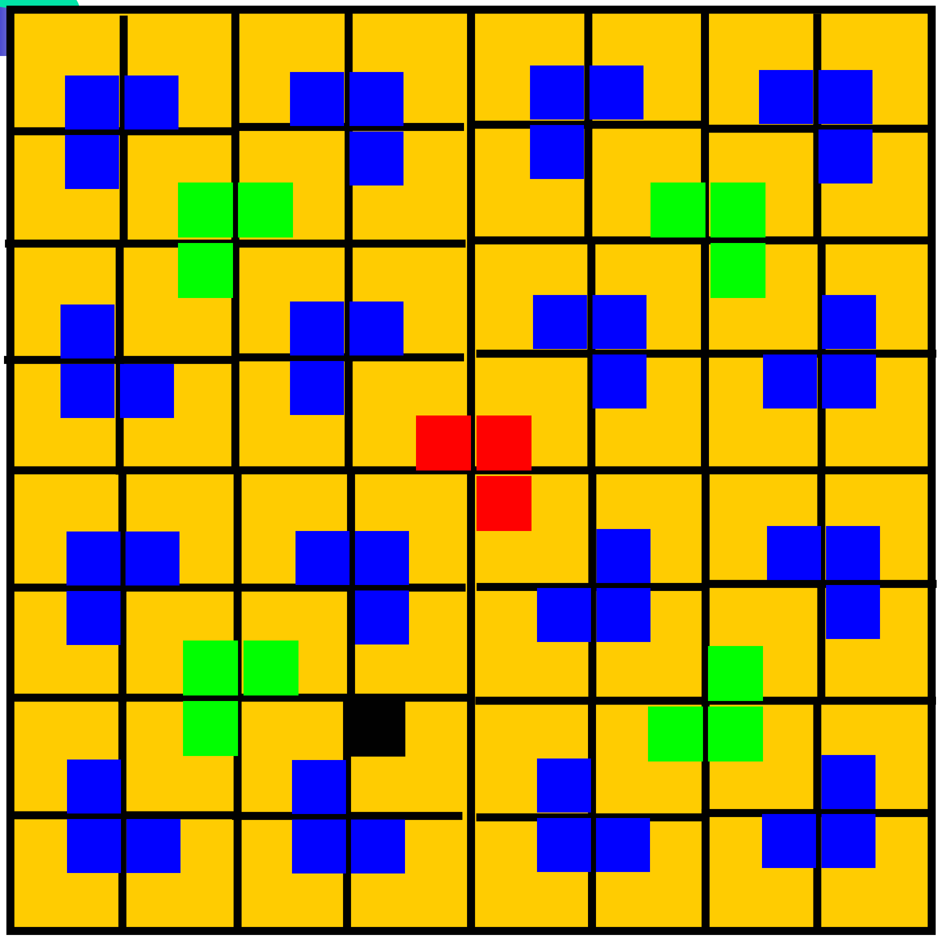
Mathematical Induction: Deficient Tiling



Mathematical Induction: Deficient Tiling



Mathematical Induction: Deficient Tiling



So, we can tile a 2^k x 2^k deficient board using our designer tiles.

What does this mean for 2^{2k} mod 3? $= \mathbf{1}$ (also do direct proof by induction)

Mathematical Induction – Exercises

Prove that chess knight (horse) can visit every square in an infinite chessboard.

Prove that n lines separate the plane into **$(n^2+n+2)/2$** regions if no two lanes are parallel and no three lines pass through the same point.

Prove that

$1/2 * 3/4 * \ldots * (2n-1)/2n < 1/\sqrt{3n}$

- can't really prove it directly, but can prove a stronger statement

  $1/2 * 3/4 * \ldots * (2n-1)/2n < 1/\sqrt{(3n+1)}$

- sometimes called **inductive loading**

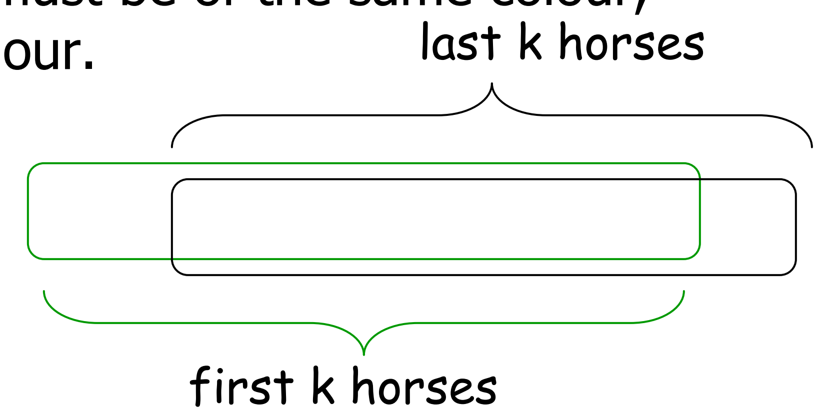
Mathematical Induction – Common Errors

What is wrong with the following "proof" of *"All horses are of the same colour"*

Base step: One horse has one colour

Induction step: Assume k horses have the same colour, we show that k+1 horses have the same colour.

Take the first k horses of our k+1 horses. By induction hypothesis they are of the same colour. The same holds for the last k horses. As these two sets overlap, they both must be of the same colour, i.e. all k+1 horses are of the same colour.



Strong Induction

P(n): a chocolate bar of size nx1 squares needs exactly n-1 breaks to break it into the basic squares.

Base case (n=1): you need 0 breaks and n-1 =0. Thus P(1) true

Inductive Step: show that $\forall(k)\ P(k) \rightarrow P(k+1)$.
Assume P(k): a chocolate bar of size kx1 squares needs exactly k-1 breaks to break it into the basic squares. and deduce P(k+1).
We break the chocolate bar into 2 pieces with sizes m and (k+1)-m.
Using the inductive hypothesis …..???

How can we use the inductive hypothesis P(k)??? We have a problem!

Use strong induction.

Strong Induction

Normal induction: To prove that **P(n)** is true for all positive integers **n**:

Base step: prove **P(1)**

Induction step: prove $\mathbf{P(n)} \rightarrow \mathbf{P(n+1)}$

Strong induction: To prove that **P(n)** is true for all positive integers **n**:

Base step: prove **P(1)**

Induction step: prove $\mathbf{P(1)} \wedge \mathbf{P(2)} \wedge \ldots \wedge \mathbf{P(n)} \rightarrow \mathbf{P(n+1)}$

Strong Induction

P(n): a chocolate bar of size nx1 squares needs exactly n-1 breaks to break it into the basic squares.

Using Strong induction:

Base case (n=1): you need 0 breaks and n-1 =0. Thus P(1) true

Inductive Step: show that $\forall(k)\ P(1) \wedge P(2) \wedge \ldots \wedge P(k) \rightarrow P(k+1)$.

We break the chocolate bar into 2 pieces with sizes m and (k+1)-m. Since both m and (k+1-m) are less than k+1, then by the strong induction hypothesis, we need m-1 breaks for the first piece and (k+1-m)-1= k-m for the second piece.

So in total we will need (m-1) + (k-m) + 1 = k = (k+1)-1 breaks for the chocolate bar with size k+1

The break used the first time

Strong Induction

Prove the following using strong induction; P(n): each postage of n cents with n at least 18s can be paid by 4c and 7c stamps.

Show that if in a round-robin tournament there exists a cycle of “player A beats player B”, then there must be a cycle of length 3.

Theorem: Every simple polygon of **n** sides can be triangulated into **n-2** triangles.

Theorem: Every triangulation of a simple polygon of $n \geq 4$ sides has at least two triangles in the triangulation with two edges on the sides of the polygon.

Theorem: Show that there is a rational number between any two real numbers.

Recursive definition

- *Recursion* is the general term for the practice of defining an object in terms of *itself*
  - or of part of itself
  - This may seem circular, but it isn't necessarily.
- An inductive proof establishes the truth of $P(n+1)$ *recursively* in terms of $P(n)$.
- There are also recursive *algorithms*, *definitions*, *functions*, *sequences*, *sets*, and other structures

Recursive definition

Recursive (Inductive) Definition of a Function:

Define **f(1)** (perhaps also **f(2), f(3)…f(k)** for some constant **k**)

Define **f(n+1)** using **f(i)** for **i** smaller then **n+1**

Example 1:

f(1) = 2, f(n+1) = 2f(n) What is the explicit value of f(n)?

Example 2:

g(1) = 1, g(n+1) = (n+1)g(n) What is the explicit value of g(n)?

We can guess the solution and then use proof by induction to do a formal check

Recursive definition

Example 1:

f(1) = 2, f(n+1) = 2f(n) What is the explicit value of f(n)?

Proof by induction that f(n) = 2^n for every n ≥1.

Base step: n=1 f(1)= 2 = 2^1. True

Inductive step: Assume that f(k) = 2^k and deduce that f(k+1)=2^{k+1}

By definition of the function f, f(k+1) = 2 f(k). By induction hypothesis f(k) = 2^k and therefore f(k+1) = 2*2^k = 2^{k+1}. This finishes the inductive step.

- **Example 2: g(1) = 1, g(n+1) = (n+1)g(n)**

Proof by induction that g(n) = n! for every n ≥1.

Base step: n=1 g(1)= 1 = 1!. True.

Inductive step: Assume that g(k) = k! and deduce that g(k+1)=(k+1)!

By definition of the function g, g(k+1) = (k+1) f(k). By induction hypothesis g(k) = k! and therefore g(k+1) = (k+1)*k!= (k+1)!. This finishes the inductive step.

Recursive definition

Example 3: **$f(n)$** – Fibonacci numbers

$f(1) = 1, f(2) = 1,$

$f(n+1) = f(n) + f(n-1)$

How fast do the Fibonacci numbers grow?

Theorem: $\forall n \geq 3, f(n) > \alpha^{(n-2)}$ where $\alpha = (1+\sqrt{5})/2$

Proof: By induction. How can we prove this?

- base step: $n = 3: f(3) = 2 > \alpha, f(4) = 3 > (3+\sqrt{5})/2 = \alpha^2$
- induction step: note that $\alpha^2 = \alpha+1$, since α is a root of $x^2-x-1 = 0.$
Therefore $\alpha^{n-1} = \alpha^2\alpha^{n-3} = (\alpha+1)\alpha^{n-3} = \alpha^{n-2}+\alpha^{n-3}$
  - by induction hypothesis, $f(n) < \alpha^{n-2}, f(n-1)<\alpha^{n-3}$, therefore as $f(n+1) = f(n)+f(n-1)$, also $f(n+1)<\alpha^{n-2}+\alpha^{n-3} = \alpha^{n-1}$

Careful with Recursive Definitions

The function defined has to be **well defined**

- it is defined for each element of its domain (often positive integers)
- it is defined unambiguously (no two different values)

Consider:

- $F(n) = 1+F(\lfloor n/2 \rfloor)$ for $n \geq 1$ and $F(1) = 1$
- $F(n) = 1+F(n-2)$ for $n \geq 1$ and $F(1) = 0$
- $F(n) = 1+F(n/3)$ for even $n \geq 3$, and $F(1) = F(2) = 1$
- $F(n) = 1+F(F(n-1))$ for $n \geq 2$ **and** $F(1) = 2$

Problems

Recursively Defined Sets and Structures

- An infinite set S may be defined recursively, by giving:
  - A small finite set of *base* elements of S.
  - A rule for constructing new elements of S from previously-established elements.
  - Implicitly, S has no other elements but these.

Example:

Let $3 \in S$, and

let if $x, y \in S$ then $x + y \in S$.

What is S?

Recursively Defined Sets and Structures

Example: Set of strings Σ^* over alphabet Σ:

Base step: the empty string $\gamma \in \Sigma^*$

Induction step: If $w \in \Sigma^*$ **and** $x \in \Sigma$ then also $wx \in \Sigma^*$

Recursively Defined Sets and Structures

Example: Let Σ be a set of symbols (the alphabet) and Σ^* be a set of strings over this alphabet. Concatenation (denoted by ".") of two strings is recursively defined as follows:

Base step: If $w \in \Sigma^*$ then, $w.\gamma = w$, where γ is the empty string

Induction step: If $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$, then $w_1.(w_2)x = (w1.w2)x$

Well-formed formulae of propositional logic:

Base step: T, F and s, where s is a propositional variable, are well-formed formulaes

Induction step: If E and F are well-formed formulae, then also $(\neg E)$, $(E \wedge F)$, $(E \vee F)$, $(E \to F)$ and $(E \leftrightarrow F)$ are well formed formulae

- how would you define well-formed arithmetic expressions?

Recursively Defined Sets and Structures

The set of full binary trees can be defined recursively:

Basic step: There is a full binary tree consisting only of a single vertex r.

Recursive step: If T1 and T2 are disjoint full binary tree, there is a full binary tree denoted by T1.T2, consisting of a root r together with edges connecting the root to each of the roots of the left subtree T1 and the right subtree T2.

We define The height h(T) of a full binary tree T recursively

Basic step: The height of the full binary tree consisting of only a root r is h(T)=0.

Recursive step: If T1 and T2 are full binary tree, then the full binary tree T=T1.T2 has height h(T)= 1 + max(h(T1), h(T2)).

Recursively Defined Sets and Structures

- Give recursive definition of
  - a rooted tree
  - a binary tree
  - internal and leaf vertices of a tree
  - length of a string

Structural Induction

Prove that every well-formed formula of propositional logic has equal number of left and right parenthesis

Base step: **T**, **F** and propositional variables do not contain parenthesis

Induction step: in every way to construct well-formed formula, the number of left and right parenthesis is the same

Structural induction of **P(x)** for every element **x** of a recursively defined set **S**:

Base step: prove **P(x)** for each element **x** of the base step definition of **S**

Induction step: for every way to construct an element **x** of **S** from elements $y_1, y_2, .. y_k$, show that $P(y_1) \wedge P(y_2) \ldots \wedge P(y_k) \rightarrow P(x)$

Structural Induction

Theorem: Let T be a full binary tree with n(T) vertices and height h(T), then $n(T) \leq 2^{h(T)+1} - 1$

Proof using structural induction:

Basis step: for the full binary tree consisting of just the root r, n(T)=1 and h(T)=0, thus $n(T)=1 \leq 2^{0+1} - 1 = 1$. Inequality is true.

Inductive step:

We assume that $n(T1) \leq 2^{h(T1)+1} - 1$ and $n(T2) \leq 2^{h(T2)+1} - 1$ for two full binary trees T1 and T2. According to the recursive formulae: $n(T)=n(T1)+n(T2)+1$ and $h(T)= 1 + \max(h(T1), h(T2))$, thus

$$
\begin{aligned}
n(T)=n(T1)+n(T2)+1 \quad & \leq (2^{h(T1)+1} - 1) + (2^{h(T2)+1} - 1) + 1 \\
& \leq (2^{h(T1)+1} + 2^{h(T2)+1}) - 1 \\
& \leq 2 * \max(2^{h(T1)+1}, 2^{h(T2)+1}) - 1 \\
& \leq 2 * 2^{\max(h(T1)+1,\, h(T2)+1)} - 1 \\
& \leq 2 * 2^{h(T)} - 1 = 2^{h(T)+1} - 1
\end{aligned}
$$

Recursive definition of height

Exercises

Let **l(x)** denote the length of a string x. Prove that **l(x.y) =l(x)+l(y).**

Every quantified formula has an equivalent one which is in prenex normal form.

Recursive Algorithms

Recursive definitions can be used to describe *algorithms.*

Typical problem solving approach: solve the problem for smaller/simpler subproblems and obtain the result from that:

```
int fact(int n)  {
   if (n == 1)  return 1;
   else return n*fact(n-1);
}

int gcd(int a, int b)   {
         if a == 0 return b;
         else return gcd(b mod a, a);
}
```

Recursion and Iteration

What about the Fibonacci sequence?

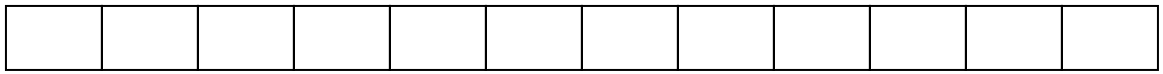
```
int fibRec(int n) {
    if (n <=2 ) return 1;
    else return fib(n-1)+fib(n-2)
}
```

can we do it iteratively?

```
int fibIter(int n) {
    int a = b = c = 1;
    for(i=2; i<n; i++) {
        c = a+b;
        a = b;
        b = c;
    }
    return c
}
```

Recursion and Iteration

Search for an element in a list



We traverse sequentially the array starting from the first cell until we find x or we finish the array

procedure *search*(a: series; i, j: integer; x: item to be found)
    if $a_i = x$ return i
    if $i = j$ return 0
    return *search*($i+1$, j, x)

No real advantage in using recursion here

location := i
while (location $\leq$ j) and (S[location] $\neq$ x) do
        location := location+1
if location > j then
        location := 0

Recursion and Iteration

When the list is already sorted, we can use a faster search: Binary search

procedure *binarySearch*(a, x, i, j)
{Find location of x in a, $\geq i$ and $< j$}
$m := \lfloor (i + j)/2 \rfloor$ {Go to halfway point.}
if $x = a_m$ **return** m
if $x < a_m \wedge i < m$ **return** {If it's to the left,}
binarySearch$(a,x,i,m-1)${Check that ½}
else if $a_m < x \wedge m < j$ **return** {If it's to right,}
binarySearch$(a,x,m+1,j)${Check that ½}
else return 0 {No more items, failure.}

Recursion and Iteration

Complexity of sequential search: $T_{SS}(n) = T(n-1) + n$

Complexity of sequential search: $T_{BS}(n) = T(n/2) + 1$

Since $1 = n/2^k$ then $k = \log n$ and thus Binary search needs $\log n + 1$ comparisons.

Sequentiel search needs at most n comparisons

Number of recursive calls

Number of comparisons par call

$$T_{BS}(n) = (k+1) * 1 = k+1$$

Recursion and Iteration

- Comparing both search algorithms:

Size	Sequential	Binary
128	128	8
1024	1024	11
1,048,576	1,048,576	21
4,294,967,296	4,294,967,296	33

Binary search is much faster however the list must be sorted