CSE 240

Logic and Discrete Math
Lecture notes on
Sequences and Induction

Weixiong Zhang
Washington University in St. Louis
http://www.cse.wustl.edu/~zhang/teaching/cse240/Spring10/index.html
Coverage

- Reading assignment – Chpt 4.1-4.5
- Sequences
  - Sums (series)
  - Products
  - Applications to Computer Science
  - Conversion to radix B
- Mathematical induction
  - Coin example
  - General rule
  - Horse mystery unveiled
  - Inductive proofs
  - Weak vs. Strong induction
- Program correctness
Sequences

- Set of elements
- Order is important
- Repetitions are allowed
- Possibly infinite
- Each one is called a term
- Index by an integer -- hence countable
- The first one (if exists) is called the initial term
- The last one (if exists) is called the final term
- Explicit/general formula specifies the elements
Representing Order in Sets

Consider two sets:
- \{Angela, Belinda\}
- \{Belinda, Angela\}

These sets are the same

What if we want to represent a specific order -- how can we do this with sets?
- \{Angela, \{Angela, Belinda\}\}

How can we recover order from this set?

In general:
- \{a_1, \{a_2, \{a_3, \ldots \{a_{n-1}, \{a_1, \ldots, a_n\} \ldots \}\right)
Examples

\[ a_k = \frac{k}{(k+1)}, \quad k > 0 \]
\[ b_i = \frac{(i-1)}{i}, \quad i > 1 \]

The same sequence -- different formulae

- Any sequence has an infinite number of explicit/general formulae

How about this sequence:

- 1, -1, 1, -1, 1, ...
Series

- A series is a sum of a sequence
- Capital Greek letter $\sum$ (sigma) is used:
  \[ \sum_{i=L}^{U} a_i = a_L + \ldots + a_U \]
  - $L$ is the lower limit
  - $U$ is the upper limit
  - Either one can be infinite
Examples

Sum $(-1)^k$

Infinite series: definitions

Arithmetic series (intro, formulae, proof)
- $a[k] = a[k-1] + d$
- [URL](http://mathworld.wolfram.com/ArithmeticSeries.html)

Geometric series (intro, formulae, proof)
- $a[k] = a[k-1] \times r$
- [URL](http://mathworld.wolfram.com/GeometricSeries.html)

Telescoping series

“Dummy variables”

Variable substitutions
Products

Use capital Greek letter $\prod$ (Pi) to denote a product:

$$U \prod_{i=L}^{U} a_i = a_L \cdot \ldots \cdot a_U$$

$L$ is the lower limit
$U$ is the upper limit
Either one can be infinite
Properties

- Distributive law
  - Sums
  - Products

- Multiplication by a constant
Applications to CS

Iterative constructs (loops)

Example:

```c
for (int I=L; I<=U; I++)
{
    Cout << I;
}
```

Arrays:

- `a[I]=a[I-1]+d;`
- `a[I]=a[I-1]*d;`
Conversion to radix B

Input:  \( N \)

Output:  \( N \) in \( B \)-radix

Algorithm:

- \( K = 0 \)
- Until \( N \neq 0 \) do
  - \( D[K] = N \mod B \)
  - \( K = K + 1 \)
  - \( N = N \div B \)
- End do

\( N = B^k \times D[k] + B^{k-1} \times D[k-1] + \ldots + D[0] \)
Examples

Factorial
- http://mathworld.wolfram.com/Factorial.html
- definition
- number of permutations of $n$ elements
- Stirling’s approximation
- binomial coefficients ($n$ choose $m$)
Mathematical Induction

First a puzzle:

What monetary amounts can one not represent with 2 and 5 cent coins?

Answer: 1 and 3 only

Proofs:

- Using remainder properties
- Using mathematical induction
General Principle

Mathematical induction as a derivation rule:

Premises:
- \( P(a) \) holds
- For any \( k \geq a \) if \( P(k) \) holds then \( P(k+1) \) holds

Conclusion:
- For any \( k \geq a \) \( P(k) \) holds

Basis step
Inductive step
Inductive hypothesis
Coin example

\[ P(n) = "n\ cents\ can\ be\ made\ up\ with\ 2\ and\ 5\ cent\ coins" \]

\[ P(4) \text{ holds } (a=4) \text{ (basis step)} \]

If \( k \) can be made up with 2 and 5 coins then \( k+1 \) can be as well \((\text{inductive step})\)
Theorem:
$2^n < n!$ for $n > 3$

Proof:
- Hand-waving argument
- Inductive proof
Proving with Math Induction

Formulate predicate $P(x)$

$P(x) = "2^x < x!"$

**Prove** the two statements:

- **Basis step:** $P(a)$
- **Inductive step:** $\forall k \geq a [ P(k) \Rightarrow P(k+1)]$

Declare that the following statement holds by the principle of mathematical induction:

- $\forall n \geq a [ P(n) ]$
  - $\forall k \geq 4 \ 2^k < k! \Rightarrow 2^{k+1} < (k+1)!$
  - $\forall n \geq 4 \ 2^n < n!$
Proof Steps

Prove the two statements:

- Basis step: \( P(a) \)
- Inductive step: \( \forall k \geq a \ [ P(k) \Rightarrow P(k+1)] \)

Existential statement
- Need to find such an \( a \)

Universal statement
- Direct proof:
  - consider an arbitrary specific \( k \) and prove the statement \( P(k) \Rightarrow P(k+1) \) for it
- Indirect proof
Equivalence

The principle of mathematical induction is equivalent to the following property:

Any set of integers $S$ satisfying:
- $a \in S$
- $\forall k \geq a \ [ k \in S \Rightarrow k+1 \in S ]$

Contains all integers greater or equal $a$
Arithmetic Series

Before we used a hand-waving argument

http://mathworld.wolfram.com/ArithmeticSeries.html

Now let’s formally prove it:

\[ \sum_{i=1}^{n} k = 1 + \ldots + n = \frac{1}{2} n (1+n) \]
Geometric Series

http://mathworld.wolfram.com/GeometricSeries.html

Again, we used a hand-waving solution before

Let’s do a formal inductive proof now…
Set \( P(n) = \text{“in any group of } n \text{ horses all horses are of the same color”} \)

Basis step:
- \( a=1, P(a) \) holds

Inductive step:
- \( \forall k \geq a \ [ P(k) \Rightarrow P(k+1)] \)
- Consider a group of \( k+1 \) horses. Show \( P(k+1) \):
  - Consider 2 horses in the group: A and B
  - Take horse A out -- the rest (\( k \) of them including B) are of the same color
  - Put A back in, take B out -- the rest (\( k \) of them including A) are of the same color
  - Pick a horse in the group other than A and B: \( \text{color}(A)=\text{color}(C) \) and \( \text{color}(B)=\text{color}(C) \)
  - So all \( k+1 \) of them are of the same color
Uses of Induction

- Proving closed-form solutions
- Proving theorems in general
- Proving invariants (program correctness)
Induction vs. Deduction

In philosophy and sciences in general (e.g., in Artificial Intelligence) the following definitions are often adopted:

- **Inductive inference** = generalization: observing a property for some instances and concluding it for all instances
  - NOT always logically valid/sound

- **Deductive inference:**
  - Specialization: from general to specific
  - Any logically valid argument

Thus, mathematical induction is...

- Deductive!
Weak/Natural Induction Schema

- Formula to prove
  - for $x \in \mathbb{N}$ holds $P[x]$

- Basis step:
  - Prove: $P[a]$

- Inductive step:
  - Assume $P[n]$
  - Prove: $P[n+1]$
Proof by Induction

Claim: for $n$ holds $6 \mid n^3-n$

Basis step:
- $n=0$

Inductive step:
- Assume that $6 \mid n^3-n$
- Prove that $6 \mid (n+1)^3-(n+1)$
Consider a recursive algorithm that takes an input of size \( n \), runs itself on two parts of the input (of size \( \text{floor}(n/2) \) and \( \text{ceil}(n/2) \)) and then uses \( n \) operations to combine the two results.

The algorithm performs 1 operation on an input of size 1

**What is the running time (as a closed-form formula)?**
Solution Step 1

Suppose the running time (i.e., the number of operations) is a function $T(n)$ of the input size $n$.

Then:

$$T(n) = \begin{cases} 
T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n & n > 1 \\
1 & n = 1 
\end{cases}$$
Solution Step 2

Need to find a closed-form solution to the recurrence

Easy:

- Make a clever guess:
  \[ T(2^k) = 2^k(1+k) \]

- Prove it using induction:
Proof by Induction

Claim: for $k$ holds $T(2^k) = 2^k(1+k)$

Basis step:
- $k=0$, $T(2^0) = 2^0 \times 1 = 1$

Inductive step:
- Assume that $T(2^k) = 2^k(1+k)$
- Prove that $T(2^{k+1}) = 2^{k+1}(1+k+1)$
More Running Times

Consider a recursive algorithm that takes an input of size \( n \), runs itself on two parts of the input (of size \( \text{floor}(n/2) \) and \( \text{floor}(n/3) \)) and then uses \( 2n \) operations to combine the two results.

The algorithm performs 0 operations on an input of size 0.

Can we upper-bound the running time (as a closed-form formula)?
Solution Step 1

Suppose the running time (i.e., the number of operations) is a function $T(n)$ of the input size $n$

Then:

$$T(n) = \begin{cases} 
T(\lfloor \frac{n}{2} \rfloor) + T(\lfloor \frac{n}{3} \rfloor) + 2n & n > 0 \\
0 & n = 0 
\end{cases}$$
Solution Step 2

Need to find a closed-form upper bound to the recurrence

Easy?:

- Make a *clever* guess:
  - $T(n) \leq 12n$
- Prove it using induction
Proof by Induction

Claim: for \( n \) holds \( T(n) \leq 12n \)

\[
T(n) = \begin{cases} 
T(\lfloor \frac{n}{2} \rfloor) + T(\lfloor \frac{n}{3} \rfloor) + 2n & n > 0 \\
0 & n = 0 
\end{cases}
\]

Basis step:
- \( n=0, \ T(0)=0 \leq 12n \)

Inductive step:
- Assume that \( T(n) \leq 12n \)
- Prove that \( T(n+1) \leq 12(n+1) \)
We are running into a problem since:
\[ T(n+1) = T(\lfloor (n+1)/2 \rfloor) + T(\lfloor (n+1)/3 \rfloor) + 2(n+1) \]

And our induction hypothesis is:
\[ T(n) \leq 12n \]

...which says nothing about:
\[ T(\lfloor (n+1)/2 \rfloor) \text{ or } T(\lfloor (n+1)/3 \rfloor) \]

We are stuck!
Let’s go back and take another look at our induction step

Inductive step:
- Assume that for a certain $n$ this holds: $T(n) \leq 12n$
- Then prove that $T(n+1) \leq 12(n+1)$

Our induction hypothesis $T(n) \leq 12n$ is just not powerful enough...

How about this instead:
- for $i \leq n$ holds $T(i) \leq 12i$
Yes!

Ok, the things worked this time

*So what was different?*

Instead of:

- Inductive step:
  - Assume $P[n]$
  - Prove $P[n+1]$

We used:

- Inductive step:
  - Assume: for $k \in \mathbb{N}$ st $k < n+1$ holds $P[k]$
  - Prove: $P[n+1]$
Strong/Complete Induction Schema

Formula to prove

- for $x \in \mathbb{N}$ holds $P[x]$

Basis step:

- Prove: $P[0]$

Inductive step:

- Assume for $k \in \mathbb{N}$ st $k < n+1$ holds $P[k]$ 
- Prove: $P[n+1]$
Strong Inductive Step Detailed

Prove: for \( n \in \mathbb{N} \) holds
\[(\text{for } k \in \mathbb{N} \text{ st } k < n+1 \text{ holds } P[k]) \text{ implies } P[n+1]\]

Let \( n \) be NATURAL

Assume (for \( k \in \mathbb{N} \text{ st } k < n+1 \) holds \( P[k] \))

Consider \( P[n+1] \)

Express it in terms of [several] \( P[j] \) such that \( j < n+1 \)

Use the induction assumption on the \( P[j] \)'s

Combine them together

Thus \( P[n+1] \)

Conclude (for \( k \in \mathbb{N} \text{ st } k < n+1 \) holds \( P[k] \)) implies \( P[n+1] \)

Therefore for \( n \in \mathbb{N} \) holds
\[(\text{for } k \in \mathbb{N} \text{ st } k < n+1 \text{ holds } P[k]) \text{ implies } P[n+1]\]
Another example

Suppose the running time (i.e., the number of operations) is a function $T(n)$ of the input size $n$.

Suppose:

$$T(n) = \begin{cases} 
4T([\frac{n}{2}]) + n^3 & n > 0 \\
0 & n = 0 
\end{cases}$$
Solution Step 2

Need to find a closed-form upper bound to the recurrence

Easy:

- Make a *clever* guess:
  - $T(n) \leq 2n^3$
- Prove it using *strong* induction
Strong Induction Proof

Formula to prove
- for $x \in \mathbb{N}$ holds $T(x) \leq 2x^3$

Basis step:
- Prove: $T(0) = 0 \leq 2 \times 0^3 = 0$

Inductive step:
- Assume: for $k \in \mathbb{N}$ st $k < n$ holds $T(k) \leq 2k^3$
- Prove: $T(n) \leq 2n^3$

\[
T(n) = \begin{cases} 
4T(\lfloor \frac{n}{2} \rfloor) + n^3 & n > 0 \\
0 & n = 0 
\end{cases}
\]
Yet another example

Suppose the running time (i.e., the number of operations) is a function $T(n)$ of the input size $n$

Suppose:

$$T(n) = \begin{cases} 
2 \cdot T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + n & n > 1 \\
0 & n = 1 
\end{cases}$$
A couple of examples

So let’s plug some numbers in:

- \( T(8) = 2 \times T(4) + 8 \)
- \( T(4) = 2 \times T(2) + 4 \)
- \( T(2) = 2 \times T(1) + 2 \)
- \( T(1) = 0 \)

Going back up:

- \( T(2) = 2 \)
- \( T(4) = 8 \)
- \( T(8) = 24 \)
Déjà Vu?

Haven’t we seen this already?

Didn’t we prove that:

\[ T(2^k) = 2^k(1+k) \]

Using weak mathematical induction?

Let’s try it then:

\[ T(8) = T(2^3) = 8 \times (3+1) = 32 \]

Previous slide: \( T(8) = 24 \)?

Where is the problem?
Mystery Unveiled

“Little details” can have a dramatic impact

Formula $T(2^k) = 2^k(k+1)$ was derived for this $T$:

$$T(n) = \begin{cases} 
T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + n & n > 1 \\
1 & n = 1 
\end{cases}$$

However, now we are dealing with a different $T$:

$$T(n) = \begin{cases} 
2 \cdot T(\lfloor n/2 \rfloor) + n & n > 1 \\
0 & n = 1 
\end{cases}$$
The new $T$

So let’s do the new function $T$:

$$T(n) = \begin{cases} 
2 \cdot T(\lfloor \frac{n}{2} \rfloor) + n & n > 1 \\
0 & n = 1 
\end{cases}$$

Anyone to guess a closed-form solution for $T(2^k)$?

Sure:
- $T(2^k) = 2^k \cdot k$

Will we need strong or weak induction?
What about a general \( n \)?

So we just proved \( T(2^k) = 2^k k \)

But what we really want is a closed form solution for \( T(n) \) where \( n \) is any natural number (not necessarily a power of 2)

How do we do this?

Let’s look at we got:

Suppose \( n = 2^k \) then \( T(n) = n k \)

How do we express \( k \) from \( n \)?

Eureka!

- \( k = \log_2 n \)
- \( T(n) = n \log_2 n \)
Let’s test it…

How about \( T(7) \):

\[
T(7) = 2 \times T(\lfloor 7/2 \rfloor) + 7 = 2 \times T(3) + 7
\]

\[
T(3) = 2 \times T(\lfloor 3/2 \rfloor) + 3 = 2 \times T(1) + 3
\]

\( T(1) = 0 \)

Going back up:

\( T(3) = 3 \)

\( T(7) = 2 \times 3 + 7 = 13 \)

Our groovy new formula, however, tells us:

\( T(7) = 7 \times \log_2 7 \approx 19.65145 \)

Oh-oh…
Another mystery resolved

Recall, that we “derived” our wonderful formula \( T(n) = n \log_2 n \) only for \( n = 2^k \)

That is when \( n \) can be divided by 2 on and on without the \( \text{floor()} \) kicking in

For non-power-of-2 \( n \)’s our formula seems to overestimate (e.g., 19 vs. 13)

Why?

Because \( \text{floor}(n/2) \) decreases the number faster than \( n/2 \) by itself
So?

So, can we still get any mileage out of our dear $T(n) = n \log_2 n$ formula?

Sure!

It is simply an upper bound for the general case.

In other words:

$T(n) \leq n \log_2 n$
Strong Induction Proof

Formula to prove

for \( x \in \mathbb{N} \) st \( x > 0 \) holds \( T(x) \leq x \log_2 x \)

Basis step:

Prove: \( T(1) = 0 \leq 1 \log_2 1 = 0 \)

Inductive step (\( n > 1 \)):

Assume: for \( k \in \mathbb{N} \) st \( 0 < k < n \) holds \( T(k) \leq k \log_2 k \)

Prove: \( T(n) \leq n \log_2 n \)
Code Correctness

- Millions of programmers code away daily...
- How do we know if their code works?
Where is CSE 240?

How do we find such bugs in software?
- Tracing
- Debug statements
- Test cases
- Many software testers working in parallel...

All of that had been employed in the previous cases
Yet the disasters occurred...
CSE240 : Program Correctness

- Logic: means to prove correctness of software
- Sometimes can be semi-automated
- Can also verify a provided correctness proof
Assertions

Every operator/step/statement in code/algorithm is annotated with pre- and post-conditions.

They are called assertions.

Technically they are formulae in predicate logic.

The pre-conditions of the first statement are the pre-conditions of the entire algorithm.

The post-conditions of the last statement are called the post-conditions of the algorithm.
Program Correctness

Therefore, proving that a piece of code is correct boils down to:

Proving that if the pre-conditions of the algorithm ($P$) are satisfied then
- The algorithm halts (eventually)
- The post-conditions ($Q$) will be satisfied upon termination

In other words, we need to show that $P \Rightarrow Q$ upon a necessary termination
Reality

Note that this formulation doesn’t say anything about:

- What happens if the inputs are wrong (e.g., will there be a warning issued or will the system quietly crash?)
- The running time
- The resource consumption
- The any-timeness nature of the algorithm

All of these are important in practice but are left out for now for simplicity
A simple example

Statement:
- \( K = 0; \)

Pre-conditions:
- \( K \) is an integer variable
- allocated in the memory space
- writeable

Post-conditions:
- \( K \) is an integer variable
- allocated in the memory space
- writeable
- the value of \( K \) is 0
Case statement

Consider a case statement:

- `Switch(K)`
  - `Case (P_1(K)) : ... ;`
  - `Case (P_2(K)) : ... ;`
  - ...
  - `Case (P_n(K)) : ... ;`
- `End switch`

How do we prove its correctness?
Case Statement

Prove that given the pre-conditions

- All tests $P_i(k)$ are computable
- At least one of them evaluates to true
- No matter which case was executed, the post-conditions will hold

Example:

- `Switch(a, b)`
  - Case $(a>b)$ : $m = a$
  - Case $(a<=b)$ : $m = b$
- End switch

- Pre-conditions: $a,b$ are integer variables, allocated, and readable
- $a>b$ and $a<=b$ are computable
- Either $a>b$ or $a<=b$ will always be true
- Post-conditions ($m=max\{a,b\}$) hold
Loop Correctness

To prove that a loop is correct we need to show that whenever loop’s pre-conditions hold it will terminate and its post-condition will necessarily hold.

Loops often imply a variable (and sometimes unknown a priori) number of iterations.

Therefore, we will have a bit more involved procedure for proving loop correctness.
Anatomy of a Loop

Consider a while loop in our pseudo-language

- While (G)
  - […statements…]
- End while

What do we need to prove?

- Given the pre-conditions are met, the loop will terminate after a finite number of iterations
- The post-conditions will be met
Tools we need

To prove just that we will need some tools:

- Pre-conditions
- Loop invariant
- Loop guard
- Post-conditions

Let’s look at them in more detail
Loop Guard

Loop guard (G) is a Boolean expression.
That is: a statement/formulae in
predicate logic over some variables.
As long as it evaluates to true the loop
will continue its execution.

Thus, we need to prove:
- G can always be evaluated provided the
  loop pre-conditions and previous iterations
- Eventually, G will necessarily become false
Consider this loop:

- **Pre-conditions:** int N,K; N>0; K=1;
- **While (N>0)**
  - N=N-1;
  - K=2*K;
- **End while**

Need to prove:
- Guard **N>0** can always be evaluated
- It will turn **false** after a finite number of iterations
Anatomy of a Loop

Consider a while loop in our pseudo-language

- **While (G)**
  - [...statements...]
- **End while**

What do we need to prove?

- Given the pre-conditions are met, the loop will terminate after a finite number of iterations
- The post-conditions will be met
Proving Post-conditions

To prove that the post-conditions are met the following technique is often used:

- Identify a property that holds at every iteration \( m \) of the loop
- Call it invariant: \( I(m) \)
  - \( I(m) \) implies \( I(m+1) \) as long as \( G(m) \) holds
- Prove that after the least number of iterations \( M \) that make \( G \) false, \( I(M) \) will imply the post-conditions
Back to our example

What are the post conditions?

- **Pre-conditions**: int N, K; N>0; K=1;
- **While (N>0)**
  - N=N-1;
  - K=2*K;
- **End while**
- **Post-conditions**: K=2^N

Ok, so we need an invariant I such that I(M) will imply the post-conditions

- M is the least number of iterations leading to G(M)=false

Can anyone think of such I?
Ok, so $I(m) \iff (K=2^m)$ seems to work

Let’s prove it:

- $I(m)$ is indeed an invariant since as long as $G(m)$ holds, $I(m) \Rightarrow I(m+1)$

- When the iteration counter $m$ hits the first $M$ such that $G(M)$ turns false (i.e., the loop exits), $I(M)$ will imply the postconditions:
  - $K=2^N$
Anatomy of a Loop

Consider a while loop in our pseudo-language

While (G)
[...statements...]
End while

What do we need to prove?

Given the pre-conditions are met, the loop will terminate after a finite number of iterations

The post-conditions will be met
Further Info

See Example 4.5.2 for slow multiplication

See Theorem 4.5.1 for a justification of the method we just went over

Additional notes on program correctness will be posted on the web