Introduction to Automata

The methods and the madness

What is the study of Automata Theory?

- The study of abstract computing devices, or “machines.”
- Days before digital computers
  - What is possible to compute with an abstract machine
  - Seminal work by Alan Turing
- Why is this useful?
  - Direct application to creating compilers, programming languages, designing applications.
  - Formal framework to analyze new types of computing devices, e.g. biocomputers or quantum computers.
  - Develop mathematically mature computer scientists capable of precise and formal reasoning!
- 5 major topics in Automata Theory

Finite State Automata

- Deterministic and non-deterministic finite state machines
- Regular expressions and languages.
- Techniques for identifying and describing regular languages; techniques for showing that a language is not regular. Properties of such languages.

Context-Free Languages

- Context-free grammars, parse trees
- Derivations and ambiguity
- Relation to pushdown automata. Properties of such languages and techniques for showing that a language is not context-free.

Turing Machines

- Basic definitions and relation to the notion of an algorithm or program.
- Power of Turing Machines.

Undecidability and Complexity

- Undecidability
  - Recursive and recursively enumerable languages.
  - Universal Turing Machines.
  - Limitations on our ability to compute; undecidable problems.
- Computational Complexity
  - Decidable problems for which no sufficient algorithms are known.
  - Polynomial time computability.
  - The notion of NP-completeness and problem reductions.
  - Examples of hard problems.

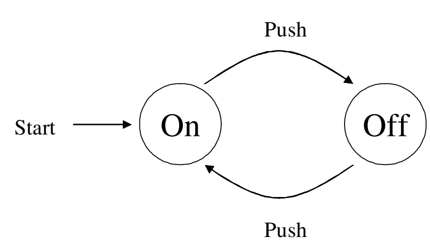
- Let's start with a big-picture overview of these 5 topics

Finite State Automata

- Automata – plural of “automaton”
  - i.e. a robot
- Finite state automata then a “robot composed of a finite number of states”
  - Informally, a finite list of states with transitions between the states
- Useful to model hardware, software, algorithms, processes
  - Software to design and verify circuit behavior
  - Lexical analyzer of a typical compiler
  - Parser for natural language processing
  - An efficient scanner for patterns in large bodies of text (e.g. text search on the web)
  - Verification of protocols (e.g. communications, security).

On-Off Switch Automaton

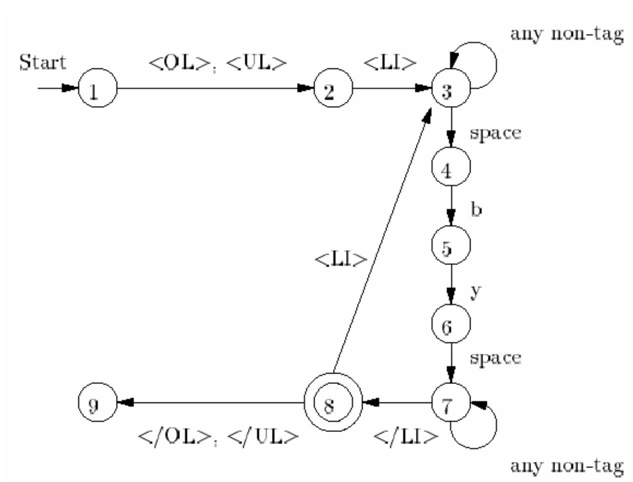
- Here is perhaps one of the simplest finite automaton, an on-off switch
- States are represented by circles. Generally we will use much more generic names for states (e.g. q1, q2). Edges or arcs between states indicate transitions or inputs to the system. The “start” edge indicates which state we start in.
- Sometimes it is necessary to indicate a “final” or “accepting” state. We’ll do this by drawing the state in double circles



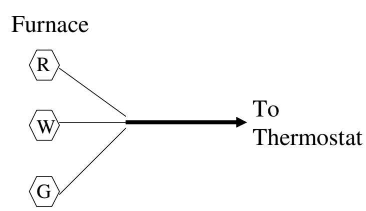
Automata Example

- Consider an automaton to parse an HTML document that attempts to identify title-author pairs in a bulleted or ordered list. This might be useful to generate a reading list of some sort automatically.
- Example:
  - Computation by Michael Sipser
- A hypothetical automaton to address this task is shown next that scans for the letters “by” inside a list item

Author Scanning Automaton



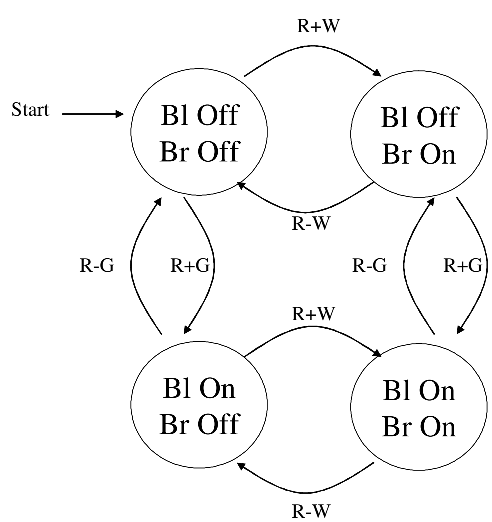
Gas Furnace Example



- The R terminal is the hot wire and completes a circuit. When R and G are connected, the blower turns on. When R and W are connected, the burner comes on. Any other state where R is not connected to either G or W results in no action.

Furnace Automaton

- Could be implemented in a thermostat



Furnace Notes

- We left out connections that have no effect
  - E.g. connecting W and G
- Once the logic in the automata has been formalized, the model can be used to construct an actual circuit to control the furnace (i.e., a thermostat).
- The model can also help to identify states that may be dangerous or problematic.
  - E.g. state with Burner On and Blower Off could overhead the furnace
  - Want to avoid this state or add some additional states to prevent failure from occurring (e.g., a timeout or failsafe)

Languages and Grammars

- Languages and grammars provide a different "view" of computing than automata
  - Often languages and grammars are identical to automata! This will be a central theme we will revisit several times.
- Consider the HTML checking automata
  - Instead of a set of states, we can view this as the problem of determining all of the strings that make up valid author/title pairs.
  - The set of all valid strings accepted by the automata makes up the Language for this particular problem

Grammar Example

- Just like English, languages can be described by grammars. For example, below is a very simple grammar:

  S → Noun Verb-Phrase
  Verb-Phrase → Verb Noun
  Noun → { Kenrick, cows }
  Verb → { loves, eats }

- Using this simple grammar our language allows the following sentences. They are "in" the Language defined by the grammar:

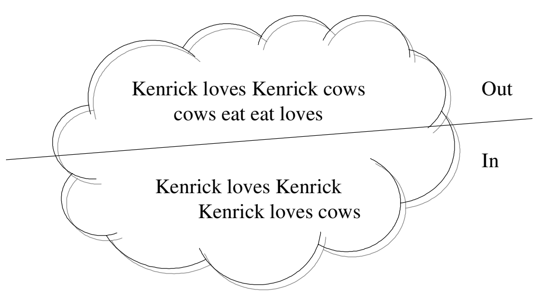
  Kenrick loves Kenrick
  Kenrick loves cows
  Kenrick eats Kenrick
  Kenrick eats cows
  Cows loves Kenrick
  Cows loves cows
  Cows eats Kenrick
  Cows eats cows

- Some sentences not in the grammar:

  Kenrick loves cows and kenrick.
  Cows eats love cows.
  Kenrick loves chocolate.

Grammars and Languages

- Later we'll see ways to go back and forth between a grammar-based definition for languages and an automata based definition
- Like a game, given a sentence (we'll call this a "string") determine if it is in or out of the Language
- Grammar provides a "cut" through the space of possible strings – will go from crude to sophisticated cuts



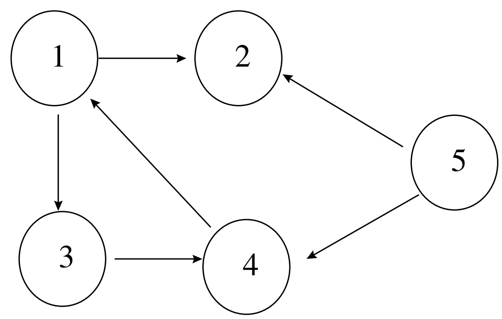
Mathematical Notions

- Skipping these topics, but they're briefly described in the textbook
  - Sets
    - Empty set, subset, union, Venn diagram, etc.
  - Sequences
  - Tuples
  - Functions and Relations
    - Mapping from Domain to Range
  - Boolean Logic

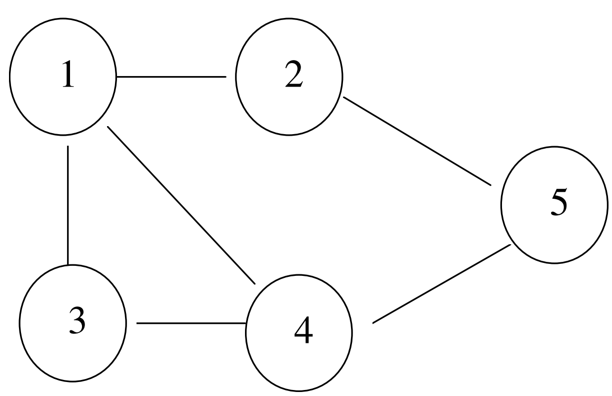
Graphs

- A graph is composed of edges *E* and vertices *V* that link the nodes together. A graph G is often denoted G=(V,E) where V is the set of vertices and E the set of edges.
- Two types of graphs:
  - Directed graphs: G=(V,E) where E is composed of ordered pairs of vertices; i.e. the edges have direction and point from one vertex to another.
  - Undirected graphs: G=(V,E) where E is composed of unordered pairs of vertices; i.e. the edges are bidirectional.

Directed Graph



Undirected Graph



Graph Terminology

- The **degree** of a vertex in an undirected graph is the number of edges that leave/enter the vertex.
- The degree of a vertex in a directed graph is the same, but we distinguish between in-degree and out-degree. Degree = in-degree + out-degree.
- A path from u to v is <u, w1, …v> and (u,w1)(w1,w2)(w2,w3)…(w_n,v)
- The running time of a graph algorithm expressed in terms of E and V, where E = |E| and V=|V|; e.g. G=O(EV) is |E| * |V|

Graph Terminology

- A **path** in a graph is a sequence of nodes connected by edges
- A graph is **connected** if every two nodes have a path between them
- A path is a **cycle** if it starts and ends in the same node
  - A **simple cycle** is one that contains at least three nodes and repeats only the first and last nodes
- A graph is a tree if it is connected and has no simple cycles
- A graph is **strongly connected** if a path connects every two nodes

Language Definitions (1)

- An **alphabet** is a finite, nonempty set of symbols. By convention we use the symbol Σ for an alphabet.
  - In the previous example, our alphabet consisted of words, but normally our alphabet will consist of individual characters.
  - Examples
    - $\Sigma = \{0,1\}$ the binary alphabet
    - $\Sigma = \{a,b, \ldots z\}$ the set of all lowercase letters

Language Definitions (2)

- **string** (or sometimes a **word**)
  - A finite sequence of symbols chosen from an alphabet. For example, 010101010 is a string chosen from the binary alphabet, as is the string 0000 or 1111.
- The **empty string** is the string with zero occurrences of symbols. This string is denoted $\boldsymbol{\varepsilon}$ and may be chosen from any alphabet.
- The power notation is used to represent multiple occurrences of a string; e.g. a^3 = aaa, a^2 = aa, etc.

Language Definitions (3)

- The **length** of a string indicates how many symbols are in that string.
  - E.g., the string 0101 using the binary alphabet has a length of 4.
  - The standard notation for a string w is to use **|w|.** For example, |0101| is 4.

Language Definitions (4)

- **Powers of an alphabet**
  - If Σ is an alphabet, we can express the set of all strings of a certain length from that alphabet by using an exponential notation.
  - Σ^k is defined to be the set of strings of length k, each of whose symbols is in Σ.
- For example, given the alphabet $\Sigma = \{0,1,2\}$ then:
  - $\Sigma^0 = \{\varepsilon\}$
  - $\Sigma^1 = \{0,1,2\}$
  - $\Sigma^2 = \{00,01,02,10,11,12,20,21,22\}$
  - $\Sigma^3 = \{000,001,002,\ldots 222\}$

- Note that Σ and Σ^1 are different. The first is the alphabet; its members are 0,1,2. The second is the set of strings whose members are the strings 0,1,2, each a string of length 1.
- By convention, we will try to use lower-case letters at the beginning of the alphabet to denote symbols, and lower-case letters near the end of the alphabet to represent strings.

Language Definitions (5)

- Set of all Strings
  - The **set of all strings** over an alphabet is denoted by Σ*****. That is:

    $$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup ...$$

  - Sometimes it is useful to exclude the empty string from the set of strings. The set of nonempty strings from the alphabet is denoted by Σ^+.

Language Definitions (6)

- To **concatenate** strings, we will simply put them right next to one another.
- Example:
  - If x and y are strings, where x=001 and y=111 then xy = 001111
  - For any string w, the equation εw = wε = w.

Formal Definition of Languages

- We have finally covered enough definitions to formally define a language!
- A Language
  - A set of strings all of which are chosen from some Σ^* is called a **language**.
  - If Σ is an alphabet and L is a subset of Σ^* then **L is a language over Σ.**
  - Note that a language need not include all strings in Σ^*.

Language Examples

1. The language of all strings consisting of n 0's followed by n 1's, for some $n \geq 0$: { ε, 01, 0011, 000111, …}
2. The set of binary numbers whose value is a prime: { 10, 11, 101, 111, …}
3. Ø is the empty language, which is a language over any alphabet.
4. {ε} is the language consisting of only the empty string. Note that this is not the same as example #3, the former has no strings and the latter has one string.

Language Definition - Problem

- A **problem** is the question of deciding whether a given string is a member of some particular language.
  - More colloquially, a problem is expressed as membership in the language.
  - Languages and problems are basically the same thing. When we care about the strings, we tend to think of it as a language. When we assign semantics to the strings, e.g. maybe the strings encode graphs, logical expressions, or integers, then we will tend to think of the set of strings as a solution to the problem.

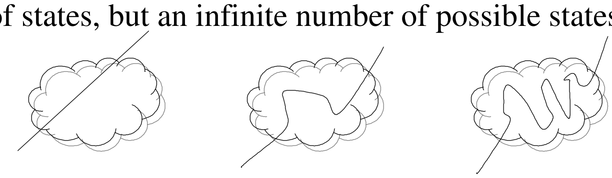
Set-Forming Notation

- A notation we will commonly use to define languages is by a "set-former":

  { w | something about w }
- The expression is read "the set of words w such that (whatever is said about w to the right of the vertical bar)."
- For example:
  - {w | w consists of an equal number of 0's and 1's }.
  - {w | w is a binary integer that is prime }
  - { 0^n1^n | n >=1 }. This includes 01, 0011, 000111, etc. but not ε
  - { 0^n1 | n>=0 }. This includes 1, 01, 001, 0001, 00001, etc.

Bigger Picture

- Finite state automata provide only a crude "cut" of $\sum^*$ to select the strings we will accept.
- Turing machines and more complex grammars provide for more sophisticated ways to define the language. One way this will be accomplished is there will no longer be a finite set of states, but an infinite number of possible states.



Crude
e.g. Simple automata

Complex
e.g. Turing Machine

Taxonomy of Complexity

Machines	Grammars/Languages
Uncomputable	
Turing Machines	Phrase Structure
Linear bounded automata	Context-Sensitive
Pushdown automata	Context-Free
Finite state automata	Regular

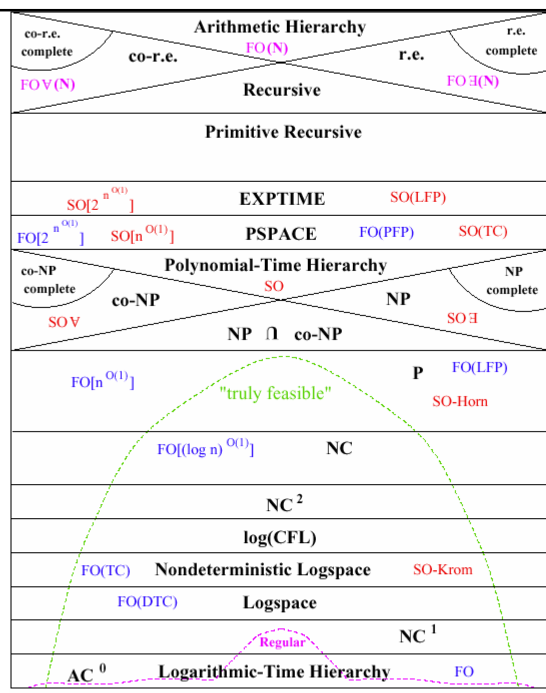
Complex

Crude

Complexity and Uncomputability

- Complexity is the study of the limits of computation. There are two important issues:
  1. Decidability. What can a computer do at all? The problems that can be solved by a computer in a realistic amount of time are called decidable. Some problems are undecidable, or only semi-decidable (e.g. membership in certain languages, must enumerate, but may be infinite)
  2. Intractability. What can a computer do efficiently? This studies the problems that can be solved by a computer using no more time than some slowly growing function of the size of the input. Typically we will take all polynomial functions to be tractable, while functions that grow faster than polynomial intractable.

Complexity Hierarchy



Introduction to Formal Proof

- In this class, sometimes we will give formal proofs and at other times intuitive "proofs"
- Mostly inductive proofs
- First, a bit about deductive proofs

Deductive Proofs

- Given a hypothesis H, and some statements, generate a conclusion C
- Sherlock Holmes style of reasoning
- Example: consider the following theorem
  - **If $x \geq 4$ then $2^x \geq x^2$**
  - Here, H is $x \geq 4$ and C is $2^x \geq x^2$
  - Intuitive deductive proof
    - Each time x increases by one, the left hand side doubles in size. However, the right side increases by the ratio $((x+1)/x)^2$. When x=4, this ratio is 1.56. As x increases and approaches infinity, the ratio $((x+1)/x)^2$ approaches 1. This means the ratio gets smaller as x increases. Consequently, 1.56 is the largest that the right hand side will increase. Since 1.56 < 2, the left side is increasing faster than the right side

Basic Formal Logic (1)

- An "If H then C" statement is typically expressed as:

  $H \Rightarrow C$ or H implies C
- The logic truth table for implication is:

H	C	$H \Rightarrow C$ (i.e. $\neg H \vee C$)
F	F	T
F	T	T
T	F	F
T	T	T

Basic Formal Logic (2)

- If and Only If statements, e.g. "If and only if H then C" means that $H \Rightarrow C$ and $C \Rightarrow H$.
- Sometimes this will be written as

  $H \Leftrightarrow C$ or "H iff C".

The truth table is:

H	C	$H \Leftrightarrow C$ (i.e. H equals C)
F	F	T
F	T	F
T	F	F
T	T	T

Modus Ponens

- **modus ponens** (Latin for "method of affirming"') can be used to form chains of logic to reach a desired conclusion.
- In other words, given:

  H⇒C and

  H

  Then we can infer C
- Example: given "If Joe and Sally are siblings then Joe and Sally are related" as a true assertion, and also given "Joe and Sally are siblings" as a true assertion, then we can conclude "Joe and Sally are related."

Modus Tollens

- **modus tollens** (Latin for "method of denying"'). This reasons backwards across the implication.
  - Cognitive psychologists have shown that under 60% of college students have a solid intuitive understanding of Modus Tollens versus almost 100% for Modus Ponens
- If we are given:

  H⇒C and

  ¬C

  then we can infer ¬H.
- For example, given: "If Joe and Sally are siblings then Joe and Sally are related" as a true assertion, and also given "Joe and Sally are not related" as a true assertion, then we can conclude "Joe and Sally are not siblings."
  - What if we are told Joe and Sally are not siblings? Can we conclude anything?

Short Exercises (1)

- If Elvis is the king of rock and roll, then Elvis lives. Elvis is the king of rock and roll. Therefore Elvis is alive. Valid or invalid?
  – This argument is valid, in that the conclusion is established (by Modus ponens) if the premises are true. However, if you consider the first premise to be false (unless you live in Vegas) then the conclusion is false.

Short Exercises (2)

- If the stock market keeps going up, then I'm going to get rich. The stock market isn't going to keep going up. Therefore I'm not going to get rich. Valid or invalid?
  – This argument is invalid, specifically an inverse error. Its form is from $\neg H$ and infer $\neg C$. This yields an inverse error.

Short Exercises (3)

- If New York is a big city, then New York has lots of people. New York has lots of people. Therefore New York is a big city. Valid or invalid?
  - This argument is invalid, even though the conclusion is true. We are given $H \Rightarrow C$ and given C. This does not mean that $C \Rightarrow H$ so we can't infer H is true.

Proof by Contradiction

- Suppose that we want to prove H and we know that C is true. Instead of proving H directly, we may instead show that assuming $\neg H$ leads to a contradiction. Therefore H must be true.
- Example:
  - A large sum of money has been stolen from the bank. The criminal(s) were seen driving away from the scene. From questioning criminals A, B, and C we know:
    - No one other than A, B, or C were involved in the robbery.
    - C never pulls a job without A
    - B does not know how to drive
  - Turned into logical statements
    1. $A \vee B \vee C$ A, B, or C is guilty
    2. $C \Rightarrow A$ If C is guilty, A is also guilty
    3. $B \Rightarrow (A \vee C)$ If B is guilty, A or C is guilty
  - Is A innocent or guilty? Let's assume that A is innocent, i.e.:
    i. $\neg A$
    ii. From $\neg A$ and #2 using modus tollens, we can infer $\neg C$
    iii. We thus have $\neg A \wedge \neg C$, which by De Morgan's Law is logically equivalent to $\neg(A \vee C)$
    iv. From $\neg(A \vee C)$ and #3 using modus tollens, we can infer $\neg B$
    v. We now have $\neg A$ and $\neg B$ and $\neg C$ which contradicts assumption #1! A is guilty.

Proof by Contrapositive

- Proof by contrapositive takes advantage of the logical equivalence between "H implies C" and "Not C implies Not H".
- For example, the assertion "If it is my car, then it is red" is equivalent to "If that car is not red, then it is not mine".
- To prove "If P, Then Q" by the method of contrapositive means to prove "If Not Q, Then Not P".

Contrapositive Example

- An integer x is called even (respectively odd) if there is another integer k for which
  $x = 2k$ (respectively $2k+1$).
- Two integers are said to have the same parity if they are both odd or both even.
- Theorem. If x and y are two integers for which $x+y$ is even, then x and y have the same parity

Contrapositive Example

- Proof of the theorem
  - The contrapositive version of this theorem is "If x and y are two integers with opposite parity, then their sum must be odd."
  - Assume x and y have opposite parity.
    - Since one of these integers is even and the other odd, there is no loss of generality to suppose x is even and y is odd.
  - Thus, there are integers k and m for which x = 2k and y = 2m+1. Then, we compute the sum x+y = 2k + 2m + 1 = 2(k+m) + 1, which is an odd integer by definition.

Contrapositive vs. Contradiction

- Both methods somewhat similar, but different.
- In contrapositive, we assume ¬C and prove ¬H, given H⇒C.
  - The method of Contrapositive has the advantage that your goal is clear: Prove Not H.
- In the method of Contradiction, your goal is to prove a contradiction, but it is not always clear what the contradiction is going to be at the start.
  - Indeed, one may never be found (and will never be found if the hypothesis is false).

Proof by Induction

- Essential for proving recursively defined objects
- We can perform induction on integers, automata, and concepts like trees or graphs.
- To make an inductive proof about a statement S(X) we need to prove two things:
  1. Basis: Prove for one or several small values of X directly.
  2. Inductive step: Assume S(Y) for Y “smaller than" X; then prove S(X) using that assumption.

Familiar Induction Example?

- For all n ≥ 0, prove that:

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

- First prove the basis. We pick n=0. When n=0, there is a general principle that when the upper limit (0) of a sum is less than the lower limit (1) then the sum is over no terms and therefore the sum is 0. That is:

$$\sum_{i=1}^{0} i = 0$$

Familiar Induction Example

- Next prove the induction. Assume n ≥ 0. We must prove that the theorem implies the same formula when n is larger. For integers, we will use n+1 as the next largest value. This means that the formula should hold with n +1 substituted for n:

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

$$= \frac{n^2 + 3n + 2}{2}$$

- This should equal what we came up with previously if we just add on an extra n+1 term:

$$\sum_{i=1}^{n+1} i = \left(\sum_{i=1}^{n} i\right) + (n+1)$$

Familiar Induction Example

- Continued: $\sum_{i=1}^{n+1} i = \left(\sum_{i=1}^{n} i\right) + (n+1)$

$$\left(\sum_{i=1}^{n} i\right) + (n+1)$$

$$= \frac{n(n+1)}{2} + (n+1)$$

$$= \frac{n^2 + n}{2} + \frac{2n+2}{2}$$

$$= \frac{n^2 + 3n + 2}{2}$$

This matches what we got from the inductive step, and the proof is complete.

Second Induction Example

- If $x \geq 4$ then $2^x \geq x^2$
- Basis: If $x=4$, then 2^x is 16 and x^2 is 16. Thus, the theorem holds.
- Induction: Suppose for some $x \geq 4$ that $2^x \geq x^2$. With this statement as the hypothesis, we need to prove the same statement, with x+1 in place of x: $2^{(x+1)} \geq (x+1)^2$

Second Induction Example

- $2^{(x+1)} \geq (x+1)^2$? (i)
- Rewrite in terms of S(x)
  - $2^{(x+1)} = 2*2^x$
  - We are assuming $2^x \geq x^2$
  - So therefore $2^{(x+1)} = 2*2^x \geq 2x^2$ (ii)
- Substitute (ii) into (i)
  - $2x^2 \geq (x+1)^2$
  - $2x^2 \geq (x^2+2x+1)$
  - $x^2 \geq 2x+1$
  - $x \geq 2 + 1/x$
  - Since x >=4, we get some value >=4 on the left side. The right side will equal at most 2.25 and in fact gets smaller and approaches 2 as x increases. Consequently, we have proven the theorem to be true by induction.