CSCI 2670 Introduction to Theory of Computing

Lecture Note 1
Introduction and Review

January 6, 2015
Tentative Schedule

1. Review (less than one week)
   - Chapter 0: set, function, string, language, theorem, proof

2. Automata and Languages (7-8 weeks)
   - Chapter 1: regular language, finite automata, nondeterminism, regular expression (3-4 weeks)
   - The first midterm exam

   - Chapter 2: context-free grammar, context-free language, push-down automata, (3-4 weeks)
   - The midterm second exam
Tentative Schedule

1. Review (less than one week)
   Chapter 0: set, function, string, language, theorem, proof
Tentative Schedule

1. Review (less than one week)
   
   Chapter 0: set, function, string, language, theorem, proof

2. Automata and Languages (7-8 weeks)
Tentative Schedule

1. Review (less than one week)
   Chapter 0: set, function, string, language, theorem, proof

2. Automata and Languages (7-8 weeks)
   Chapter 1: regular language, finite automata, nondeterminism, regular expression (3-4 weeks)
1. Review (less than one week)
   Chapter 0: set, function, string, language, theorem, proof

2. Automata and Languages (7-8 weeks)
   Chapter 1: regular language, finite automata, nondeterminism, regular expression (3-4 weeks)

   The first midterm exam
Tentative Schedule

1. Review (less than one week)
   
   Chapter 0: set, function, string, language, theorem, proof

2. Automata and Languages (7-8 weeks)
   
   Chapter 1: regular language, finite automata, nondeterminism, regular expression (3-4 weeks)
   
   The first midterm exam
   
   Chapter 2. context-free grammar, context-free language, push-down automata, (3-4 weeks)
Tentative Schedule

1. Review (less than one week)
   Chapter 0: set, function, string, language, theorem, proof

2. Automata and Languages (7-8 weeks)
   Chapter 1: regular language, finite automata, nondeterminism, regular expression (3-4 weeks)
   The first midterm exam
   Chapter 2. context-free grammar, context-free language, push-down automata, (3-4 weeks)
   The midterm second exam
Tentative Schedule

3. Computability Theory (3-4 weeks)
   - Chapter 3: Turing machine, Chomsky hierarchy
   - Chapter 4: decidable language, Halting problem
   - Chapter 5: undecidable language, reduction

4. Introduction to Complexity Theory (1-2 weeks)
   - Chapter 7: time complexity, P, NP-completness
   - Chapter 9: intractability

The final exam
3. Computability Theory (3-4 weeks)

- Chapter 3: Turing machine, Chomsky hierarchy,
- Chapter 4: decidable language, Halting problem,
- Chapter 5: undecidable language, reduction
Tentative Schedule

3. Computability Theory (3-4 weeks)

   Chapter 3: Turing machine, Chomsky hierarchy,
   Chapter 4: decidable language, Halting problem,
   Chapter 5: undecidable language, reduction

4. Introduction to Complexity Theory (1-2 weeks)

   Chapter 7: time complexity, P, NP-completeness
   chapter 9: intractability
3. Computability Theory (3-4 weeks)
   - Chapter 3: Turing machine, Chomsky hierarchy,
   - Chapter 4: decidable language, Halting problem,
   - Chapter 5: undecidable language, reduction

4. Introduction to Complexity Theory (1-2 weeks)
   - Chapter 7: time complexity, P, NP-completeness
   - Chapter 9: intractability

The final exam
Motivations

1. Foundation of computer science: the theory existed before the first computer model.
2. Theory and techniques for core CS sub-disciplines: programming language and compiler design, text processing, algorithm design, complexity theory, parallel computing, etc.
3. Elegant, simple way to think about computation: fundamental issues remain regardless advanced technologies.
Motivations

Why theory of computing?
Motivations

Why theory of computing?

1. foundation of computer science
   the theory existed before the first computer model
Motivations

Why theory of computing?

1. foundation of computer science
   the theory existed before the first computer model

2. theory and techniques for core CS sub-disciplines
   programming language and compiler design, text processing,
   algorithm design, complexity theory, parallel computing, etc.
Motivations

Why theory of computing?

1. foundation of computer science
   the theory existed before the first computer model

2. theory and techniques for core CS sub-disciplines
   programming language and compiler design, text processing,
   algorithm design, complexity theory, parallel computing, etc.

3. elegant, simple way to think about computation
   fundamental issues remain regardless advanced technologies
Motivations

Why theory of computing?

1. foundation of computer science
   the theory existed before the first computer model

2. theory and techniques for core CS sub-disciplines
   programming language and compiler design, text processing,
   algorithm design, complexity theory, parallel computing, etc.

3. elegant, simple way to think about computation
   fundamental issues remain regardless advanced technologies

4. emerging applications
   internet search, bio-medical sciences
   non-traditional computation models
Objectives
Objectives

What are the goals of this course?
Objectives

What are the goals of this course?

Summarized as the Chomsky Hierarchy and extension

<table>
<thead>
<tr>
<th>Model</th>
<th>Languages/Grammar</th>
<th>What are they?</th>
</tr>
</thead>
</table>
Objectives

What are the goals of this course?

Summarized as the Chomsky Hierarchy and extension

<table>
<thead>
<tr>
<th>Model</th>
<th>Languages/Grammar</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite automata</td>
<td>regular</td>
<td>constant memory</td>
</tr>
</tbody>
</table>
Objectives

What are the goals of this course?

Summarized as the Chomsky Hierarchy and extension

<table>
<thead>
<tr>
<th>Model</th>
<th>Languages/Grammar</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite automata</td>
<td>regular</td>
<td>constant memory</td>
</tr>
<tr>
<td>push-down FA</td>
<td>context-free</td>
<td>an additional stack</td>
</tr>
</tbody>
</table>
Objectives

What are the goals of this course?

Summarized as the **Chomsky Hierarchy** and extension

<table>
<thead>
<tr>
<th>Model</th>
<th>Languages/Grammar</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite automata</td>
<td>regular</td>
<td>constant memory</td>
</tr>
<tr>
<td>push-down FA</td>
<td>context-free</td>
<td>an additional stack</td>
</tr>
<tr>
<td>linear-bounded TMs</td>
<td>context-sensitive</td>
<td>linear memory</td>
</tr>
</tbody>
</table>
### Objectives

**What are the goals of this course?**

Summarized as the **Chomsky Hierarchy** and extension

<table>
<thead>
<tr>
<th>Model</th>
<th>Languages/Grammar</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite automata</td>
<td>regular</td>
<td>constant memory</td>
</tr>
<tr>
<td>push-down FA</td>
<td>context-free</td>
<td>an additional stack</td>
</tr>
<tr>
<td>linear-bounded TMs</td>
<td>context-sensitive</td>
<td>linear memory</td>
</tr>
<tr>
<td>Turing machines</td>
<td>decidable</td>
<td>unlimited memory</td>
</tr>
</tbody>
</table>
What are the goals of this course?

Summarized as the **Chomsky Hierarchy** and extension

<table>
<thead>
<tr>
<th>Model</th>
<th>Languages/Grammar</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite automata</td>
<td>regular</td>
<td>constant memory</td>
</tr>
<tr>
<td>push-down FA</td>
<td>context-free</td>
<td>an additional stack</td>
</tr>
<tr>
<td>linear-bounded TMs</td>
<td>context-sensitive</td>
<td>linear memory</td>
</tr>
<tr>
<td>Turing machines</td>
<td>decidable</td>
<td>unlimited memory</td>
</tr>
<tr>
<td>unknown</td>
<td>undecidable</td>
<td>unknown</td>
</tr>
</tbody>
</table>
# Objectives

## What are the goals of this course?

Summarized as the **Chomsky Hierarchy** and extension

<table>
<thead>
<tr>
<th>Model</th>
<th>Languages/Grammar</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite automata</td>
<td>regular</td>
<td>constant memory</td>
</tr>
<tr>
<td>push-down FA</td>
<td>context-free</td>
<td>an additional stack</td>
</tr>
<tr>
<td>linear-bounded TMs</td>
<td>context-sensitive</td>
<td>linear memory</td>
</tr>
<tr>
<td>Turing machines</td>
<td>decidable</td>
<td>unlimited memory</td>
</tr>
<tr>
<td>unknown</td>
<td>undecidable</td>
<td>unknown</td>
</tr>
<tr>
<td>Polynomial-time TMs</td>
<td>class P</td>
<td>tractable problems</td>
</tr>
</tbody>
</table>
### Objectives

What are the goals of this course?

Summarized as the **Chomsky Hierarchy** and extension

<table>
<thead>
<tr>
<th>Model</th>
<th>Languages/Grammar</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite automata</td>
<td>regular</td>
<td>constant memory</td>
</tr>
<tr>
<td>push-down FA</td>
<td>context-free</td>
<td>an additional stack</td>
</tr>
<tr>
<td>linear-bounded TMs</td>
<td>context-sensitive</td>
<td>linear memory</td>
</tr>
<tr>
<td>Turing machines</td>
<td>decidable</td>
<td>unlimited memory</td>
</tr>
<tr>
<td>unknown</td>
<td>undecidable</td>
<td>unknown</td>
</tr>
<tr>
<td>Polynomial-time TMs</td>
<td>class P</td>
<td>tractable problems</td>
</tr>
<tr>
<td>Polynomial-time NTMs</td>
<td>class NP</td>
<td>intractable</td>
</tr>
</tbody>
</table>
Some explanations for Chomsky Hierarchy

How powerful (powerless) are programs with very limited memory?

What can a program do if using only 4 variables Int X, Y, Z, W?

- can only memorize small amount of information
- what can it not do?
- cannot count!
- e.g., cannot correctly recognize long expressions like 
  \((x+20) \times ((y-z) \times (w+u)-40) \times v)\)

This problem is "context-free", while the machine model is "regular".
Some explanations for Chomsky Hierarchy

*How powerful (powerless) are programs with very limited memory?*
Some explanations for Chomsky Hierarchy

How powerful (powerless) are programs with very limited memory?

What can a program do if using only 4 variables

\texttt{Int X, Y, Z, W; ?}
Some explanations for Chomsky Hierarchy

*How powerful (powerless) are programs with very limited memory?*

What can a program do if using only 4 variables

    Int X, Y, Z, W; ?

...can only memorize small amount of information
Some explanations for Chomsky Hierarchy

*How powerful (powerless) are programs with very limited memory?*

What can a program do if using only 4 variables

```
Int X, Y, Z, W; ?
```

can only memorize small amount of information

*what can it not do?*
Some explanations for Chomsky Hierarchy

*How powerful (powerless) are programs with very limited memory?*

What can a program do if using only 4 variables

```plaintext
Int X, Y, Z, W; ?
```

can only memorize small amount of information

*what can it not do? cannot count!*
Some explanations for Chomsky Hierarchy

How powerful (powerless) are programs with very limited memory?

What can a program do if using only 4 variables

Int X, Y, Z, W; ?

can only memorize small amount of information
what can it not do? cannot count!

e.g., cannot correctly recognize long expressions like

\[(x + 20) \times (((y - z) \times (w + u) - 40) \times v)\]
How powerful (powerless) are programs with very limited memory?

What can a program do if using only 4 variables

```c
Int X, Y, Z, W; ?
```

can only memorize small amount of information

what can it not do? cannot count!

e.g., cannot correctly recognize long expressions like

```
(x + 20) \times (((y - z) \times (w + u) - 40) \times v)
```

or simply

```
( ( ( ( ) ) ) )
```
Some explanations for Chomsky Hierarchy

*How powerful (powerless) are programs with very limited memory?*

What can a program do if using only 4 variables

```
int X, Y, Z, W; //
```

can only memorize small amount of information

*what can it not do? cannot count!*

e.g., cannot correctly recognize long expressions like

\[(x + 20) \times (((y - z) \times (w + u) - 40) \times v)\]

or simply

\[( ) ( ( ( ) ( ) ) )\]

This problem is "context-free", while the machine model is "regular".
Some explanations for Chomsky Hierarchy

A stack would help!

How?

push every '(' encountered, and pop '(' for every encountered ')

stack has unlimited memory but access is in a restricted way.

e.g., recognizing strings

\textit{aa...abb...b}

(with the same number

of

'a's and

'b's)

also recognizing palindroms

\textit{xy...zz...yx}

But can strings

\textit{aa...abb...bcc...c}

be recognized with a single stack?

NO! But two stacks would work.

How?
Some explanations for Chomsky Hierarchy

A stack would help!

How?

push every '(' encountered, and pop '(' for every encountered ')' 

stack has unlimited memory but access is in a restricted way. 

e.g., recognizing strings 

aa...abb...b 

(with the same number of a's and b's) 

also recognizing palindroms 

xy...zz...yx 

But can strings 

aa...abb...bcc...c 

be recognized with a single stack? 

NO! But two stacks would work. 

How?
Some explanations for Chomsky Hierarchy

A stack would help!

How?
Some explanations for Chomsky Hierarchy

A stack would help!

How?

    push every '(' encountered, and
A stack would help!

How?

push every ‘(’ encountered, and
pop ‘(’ for every encountered ’)’
Some explanations for Chomsky Hierarchy

A stack would help!

How?

push every '(' encountered, and
pop '(' for every encountered ')

stack has unlimited memory but access is in a restricted way.
Some explanations for Chomsky Hierarchy

A stack would help!

How?

push every '(' encountered, and
pop '(' for every encountered ')

stack has unlimited memory but access is in a restricted way.

e.g., recognizing strings \texttt{aa\ldots abb\ldots b} (with the same number of a’s and b’s)
Some explanations for Chomsky Hierarchy

A stack would help!

How?

push every ’(’ encountered, and
pop ’(’ for every encountered ’)’

stack has unlimited memory but access is in a restricted way.

e.g., recognizing strings aa...abb...b (with the same number of a’s and b’s)

also recognizing palindroms xy...zz...yx
Some explanations for Chomsky Hierarchy

A stack would help!

How?

push every '(' encountered, and
pop '(' for every encountered ')

stack has unlimited memory but access is in a restricted way.

e.g., recognizing strings \texttt{aa...abb...b} (with the same number of a’s and b’s)

also recognizing palindroms \texttt{xy...zz...yx}

But can strings \texttt{aa...abb...bcc...c} be recognized with a single stack?
Some explanations for Chomsky Hierarchy

A stack would help!

How?

push every ’(’ encountered, and
pop ’(’ for every encountered ’)’

stack has unlimited memory but access is in a restricted way.

e.g., recognizing strings \textit{aa...abb...b} (with the same number of a’s and b’s)

also recognizing palindroms \textit{xy...zz...yx}

But can strings \textit{aa...abb...bcc...c} be recognized with a single stack?

\textbf{NO!} But two stacks would work.
A stack would help!

How?

push every '(' encountered, and 
pop '(' for every encountered ')

stack has unlimited memory but access is in a restricted way.

e.g., recognizing strings $aa\ldots abb\ldots b$ (with the same number
of a’s and b’s)

also recognizing palindroms $xy\ldots zz\ldots yx$

But can strings $aa\ldots abb\ldots bcc\ldots c$ be recognized with a single stack?

NO! But two stacks would work.

How?
Some explanations for Chomsky Hierarchy

What is the fundamental difference between strings 
\[ \text{aa...abb...b} \] 
\[ \text{aa...a} \text{bb...b} \text{cc...c} \] 
The way a stack works is in the "nested" (and "parallel") fashion 
\[ ((())) \] 
\[ ((()))(()()) \] 
So the pairing is 3-4, 2-5, 1-6, 8-9, and 7-10.
But 
\[ ((()))]] \] 
\[ 123456789 \] 
would need pairings 3-4, 2-5, 1-6, 6-7, 5-8. and 4-9 which involves "crossing" patterns.
Some explanations for Chomsky Hierarchy

What is the fundamental difference between strings

aa...abb...b and
Some explanations for Chomsky Hierarchy

What is the fundamental difference between strings

\[ \text{aa...abb...b and aa...abb...bcc...c ?} \]
Some explanations for Chomsky Hierarchy

What is the fundamental difference between strings

\[ aa\ldots abb\ldots b \text{ and } aa\ldots abb\ldots bcc\ldots c \, ? \]

The way a stack works is in the "nested" (and "parallel") fashion

\[ (((()))) \text{ and } \]
Some explanations for Chomsky Hierarchy

What is the fundamental difference between strings

\[ \text{aa...abb...b and } \text{aa...abb...bcc...c?} \]

The way a stack works is in the "nested" (and "parallel") fashion

\[
((())) \quad \text{and} \quad ((()))((())) \quad \text{and such}
\]

\[
123456 \quad 12345678910
\]
Some explanations for Chomsky Hierarchy

What is the fundamental difference between strings

\[aa...abb...b\] and \[aa...abb...bcc...c\]?

The way a stack works is in the "nested" (and "parallel") fashion

\[(((\textsf{()})\textsf{()})\textsf{()})\textsf{()}\] and such
\[123456\quad 12345678910\]

So the pairing is 3-4, 2-5, 1-6, 8-9, and 7-10.
Some explanations for Chomsky Hierarchy

What is the fundamental difference between strings

aa...abb...b and
aa...abb...bcc...c?

The way a stack works is in the "nested" (and "parallel") fashion

((())) and ((()))((())) and such
123456       12345678910

So the pairing is 3-4, 2-5, 1-6, 8-9, and 7-10.

But (((()]]])]]
123456789

would need pairings 3-4, 2-5, 1-6, 6-7, 5-8. and 4-9
Some explanations for Chomsky Hierarchy

What is the fundamental difference between strings

aa...abb...b and
aa...abb...bcc...c?

The way a stack works is in the "nested" (and "parallel") fashion

((())) and (((())))((())) and such
123456  12345678910

So the pairing is 3-4, 2-5, 1-6, 8-9, and 7-10.

But (((())))]]]]
123456789

would need pairings 3-4, 2-5, 1-6, 6-7, 5-8. and 4-9

which involves "crossing" patterns.
Some explanations for Chomsky Hierarchy

Intuitively, non-nested or non-crossing patterns, easy problems, can be handled with limited (finite) memory. Nested patterns, moderately hard problems, can be handled with one stack (infinite but restricted access memory). Crossing patterns, hard problems, can be handled with two stacks (infinite, random access memory). NOTE: the last class of problems is the largest that a computer can handle.
Some explanations for Chomsky Hierarchy

Intuitively,

non-nested or non-crossing patterns, easy problems, can be handled with limited (finite) memory
Some explanations for Chomsky Hierarchy

Intuitively,

non-nested or non-crossing patterns, easy problems,
can be handled with limited (finite) memory

nested patterns, moderately hard problems,
can be handled with one stack (infinite but restricted access memory)
Some explanations for Chomsky Hierarchy

Intuitively,

- non-nested or non-crossing patterns, easy problems, can be handled with limited (finite) memory

- nested patterns, moderately hard problems, can be handled with one stack (infinite but restricted access memory)

- crossing patterns, hard problems, can be handled with two stacks (infinite, random access memory)
Some explanations for Chomsky Hierarchy

Intuitively,

- non-nested or non-crossing patterns, easy problems, can be handled with limited (finite) memory
- nested patterns, moderately hard problems, can be handled with one stack (infinite but restricted access memory)
- crossing patterns, hard problems, can be handled with two stacks (infinite, random access memory)

**NOTE:** the last class of problems is the largest that computer can handle.
Some explanations for Chomsky Hierarchy

More about context-free problems
instances have nested, or/and parallel "patterns"
described as trees
straightforward recursive implementation
Additional discussion about recursion:
recursive programs can be executed with a stack
it uses more memory than those defined variables
Some explanations for Chomsky Hierarchy

More about context-free problems
Some explanations for Chomsky Hierarchy

More about context-free problems

instances have nested, or/and parallel "patterns"
Some explanations for Chomsky Hierarchy

More about context-free problems

instances have nested, or/and parallel "patterns"

handled by a single stack
Some explanations for Chomsky Hierarchy

More about context-free problems

instances have nested, or/and parallel "patterns"

handled by a single stack
described as trees
Some explanations for Chomsky Hierarchy

More about context-free problems

instances have nested, or/and parallel "patterns"

handled by a single stack
described as trees
straightforward recursive implementation
Some explanations for Chomsky Hierarchy

More about context-free problems

instances have nested, or/and parallel "patterns"

handled by a single stack
described as trees
straightforward recursive implementation

Additional discussion about recursion:
Some explanations for Chomsky Hierarchy

More about context-free problems

instances have nested, or/and parallel ”patterns”

handled by a single stack

described as trees

straightforward recursive implementation

Additional discussion about recursion:

recursive programs can be executed with a stack
Some explanations for Chomsky Hierarchy

More about context-free problems

- instances have nested, or/and parallel "patterns"
  - handled by a single stack
  - described as trees
  - straightforward recursive implementation

Additional discussion about recursion:

- recursive programs can be executed with a stack
- it uses more memory than those defined variables
Some explanations for Chomsky Hierarchy
Some explanations for Chomsky Hierarchy

More about non-context-free problems (context-sensitive)
Some explanations for Chomsky Hierarchy

More about non-context-free problems (context-sensitive)

Could be very sophisticated
Some explanations for Chomsky Hierarchy

More about non-context-free problems (context-sensitive)

Could be very sophisticated

([)]([ ][[ ]]{ })}
Some explanations for Chomsky Hierarchy

More about non-context-free problems (context-sensitive)

Could be very sophisticated

( [] ( [ ] [ ] ) )

Those pairings can be thought of as information of related objects in the input or to be computed and outputted!
Chapter 0 Introduction
Chapter 0 Introduction

Review of mathematical concepts
Chapter 0 Introduction

Review of mathematical concepts

sets, basic set operations, properties
Chapter 0 Introduction

Review of mathematical concepts
sets, basic set operations, properties
relations, functions, predicates
Chapter 0 Introduction

Review of mathematical concepts

sets, basic set operations, properties
relations, functions, predicates
strings, languages,
Chapter 0 Introduction

Review of mathematical concepts

sets, basic set operations, properties
relations, functions, predicates
strings, languages,
Boolean logic, theorem, proofs
Chapter 0 Introduction
Chapter 0 Introduction

**Set:** a collection of (related, discrete) objects
**Set**: a collection of (related, discrete) objects

elements in a set: \( x \in S \)
Chapter 0 Introduction

**Set**: a collection of (related, discrete) objects

- elements in a set: \( x \in S \)
- empty set: \( \emptyset \)
**Set**: a collection of (related, discrete) objects

- elements in a set: $x \in S$
- empty set: $\emptyset$
- cardinality of a set: $|S|$, infinite set
Chapter 0 Introduction

**Set:** a collection of (related, discrete) objects

- elements in a set: \( x \in S \)
- empty set: \( \emptyset \)
- cardinality of a set: \( |S| \), infinite set
- subset: \( A \subseteq B \), and superset, proper subset \( A \subset B \)
Chapter 0 Introduction

**Set:** a collection of (related, discrete) objects

- elements in a set: \( x \in S \)
- empty set: \( \emptyset \)
- cardinality of a set: \( |S| \), infinite set
- subset: \( A \subseteq B \), and superset, proper subset \( A \subset B \)
- complement of a set: \( \bar{S} \)
**Chapter 0 Introduction**

**Set**: a collection of (related, discrete) objects

- elements in a set: \( x \in S \)
- empty set: \( \phi \)
- cardinality of a set: \( |S| \), infinite set
- subset: \( A \subseteq B \), and superset, proper subset \( A \subset B \)
- complement of a set: \( \bar{S} \)
- union of two sets: \( A \cup B \), intersection of two sets: \( A \cap B \)
Set: a collection of (related, discrete) objects

elements in a set: \( x \in S \)
empty set: \( \phi \)
cardinality of a set: \(|S|\), infinite set
subset: \( A \subseteq B \), and superset, proper subset \( A \subset B \)
complement of a set: \( \bar{S} \)
union of two sets: \( A \cup B \), intersection of two sets: \( A \cap B \)
Cartesian product (cross product):

\[
A \times B = \{(a, b) : a \in A, b \in B\}
\]
Chapter 0 Introduction

**Set**: a collection of (related, discrete) objects

- elements in a set: $x \in S$
- empty set: $\emptyset$
- cardinality of a set: $|S|$, infinite set
- subset: $A \subseteq B$, and superset, proper subset $A \subset B$
- complement of a set: $\bar{S}$
- union of two sets: $A \cup B$, intersection of two sets: $A \cap B$
- Cartesian product (cross product): $A \times B = \{(a, b) : a \in A, b \in B\}$
- Power set: $\mathcal{P}(A) = \{B : B \subseteq A\}$, or also denoted with $2^A$
Chapter 0 Introduction

**Set:** a collection of (related, discrete) objects

- elements in a set: \( x \in S \)
- empty set: \( \emptyset \)
- cardinality of a set: \(|S|\), infinite set
- subset: \( A \subseteq B \), and superset, proper subset \( A \subset B \)
- complement of a set: \( \bar{S} \)
- union of two sets: \( A \cup B \), intersection of two sets: \( A \cap B \)

Cartesian product (cross product):

\[
A \times B = \{(a, b) : a \in A, b \in B\}
\]

Power set: \( \mathcal{P}(A) = \{B : B \subseteq A\} \), or also denoted with \( 2^A \)

**how many elements in** \( 2^A \) **?**
Chapter 0 Introduction

Relation: a subset of Cartesian product of sets
function \( f : X \rightarrow Y \), is a binary relation \( \subseteq X \times Y \) where \( X \) is domain and \( Y \) range.

e.g., \( f \) defined by \( f(1) = -3, f(2) = 2, f(3) = 4 \) is a binary relation \( R_f = \{ (1, -3), (2, 2), (3, 4) \} \)

binary relations can be many-many, many-1, 1-many, or 1-1.
a function cannot be 1-many.

1-1 function (injection): an 1-1 relation
onto function (surjection)

\( f : \forall y \in Y, \exists x \in X, (x, y) \in R_f \)

one-to-one correspondence (bijection): a both 1-1 and onto function

\( k \)-ary relation: a subset of \( A \times A \times \ldots \times A \) (\( k \) times)
Chapter 0 Introduction

**Relation**: a subset of Cartesian product of sets
Chapter 0 Introduction

**Relation**: a subset of Cartesian product of sets

**function** $f : X \rightarrow Y$, is a binary relation $\subseteq X \times Y$

where $X$ is domain and $Y$ range.
Chapter 0 Introduction

**Relation**: a subset of Cartesian product of sets

**function** $f : X \rightarrow Y$, is a binary relation $\subseteq X \times Y$ where $X$ is domain and $Y$ range.

e.g., $f$ defined by $f(1) = -3, f(2) = 2, f(3) = 4$

is a binary relation $R_f = \{(1, -3), (2, 2), (3, 4)\}$
**Relation**: a subset of Cartesian product of sets

**function** $f : X \rightarrow Y$, is a binary relation $\subseteq X \times Y$

where $X$ is domain and $Y$ range.

e.g., $f$ defined by $f(1) = -3, f(2) = 2, f(3) = 4$

is a binary relation $R_f = \{(1, -3), (2, 2), (3, 4)\}$

binary relations can be many-many, many-1, 1-many, or 1-1.
Chapter 0 Introduction

**Relation**: a subset of Cartesian product of sets

**function** $f : X \rightarrow Y$, is a binary relation $\subseteq X \times Y$
where $X$ is domain and $Y$ range.

e.g., $f$ defined by $f(1) = -3, f(2) = 2, f(3) = 4$
is a binary relation $R_f = \{(1, -3), (2, 2), (3, 4)\}$

binary relations can be many-many, many-1, 1-many, or 1-1.
a function cannot be 1-many.
Chapter 0 Introduction

Relation: a subset of Cartesian product of sets

function \( f : X \rightarrow Y \), is a binary relation \( \subseteq X \times Y \)
where \( X \) is domain and \( Y \) range.

e.g., \( f \) defined by \( f(1) = -3, f(2) = 2, f(3) = 4 \)
is a binary relation \( R_f = \{(1, -3), (2, 2), (3, 4)\} \)

binary relations can be many-many, many-1, 1-many, or 1-1.
a function cannot be 1-many.

1-1 function (injection): an 1-1 relation
Chapter 0 Introduction

**Relation**: a subset of Cartesian product of sets

**function** $f : X \to Y$, is a binary relation $\subseteq X \times Y$
where $X$ is domain and $Y$ range.

e.g., $f$ defined by $f(1) = -3$, $f(2) = 2$, $f(3) = 4$
is a binary relation $R_f = \{(1, -3), (2, 2), (3, 4)\}$

binary relations can be many-many, many-1, 1-many, or 1-1.

a function cannot be 1-many.

**1-1 function** (injection): an 1-1 relation

**onto function** (surjection) $f: \forall y \in Y, \exists x \in X, (x, y) \in R_f$
Chapter 0 Introduction

**Relation**: a subset of Cartesian product of sets

**function** $f : X \rightarrow Y$, is a binary relation $\subseteq X \times Y$
where $X$ is domain and $Y$ range.

e.g., $f$ defined by $f(1) = -3, f(2) = 2, f(3) = 4$

is a binary relation $R_f = \{(1, -3), (2, 2), (3, 4)\}$

binary relations can be many-many, many-1, 1-many, or 1-1.

a function cannot be 1-many.

**1-1 function** (injection): an 1-1 relation

**onto function** (surjection) $f$: $\forall y \in Y, \exists x \in X, (x, y) \in R_f$

**one-to-one correspondence** (bijection): a both 1-1 and onto function
Chapter 0 Introduction

**Relation**: a subset of Cartesian product of sets

**function** $f : X \rightarrow Y$, is a binary relation $\subseteq X \times Y$
where $X$ is domain and $Y$ range.

e.g., $f$ defined by $f(1) = -3, f(2) = 2, f(3) = 4$

is a binary relation $R_f = \{(1, -3), (2, 2), (3, 4)\}$

binary relations can be many-many, many-1, 1-many, or 1-1.

a function cannot be 1-many.

**1-1 function** (injection): an 1-1 relation

**onto function** (surjection) $f$: $\forall y \in Y, \exists x \in X, (x, y) \in R_f$

**one-to-one correspondence** (bijection): a both 1-1 and onto function

**$k$-ary relation**: a subset of $A \times A \times \cdots \times A$ ($k$ times)
Chapter 0 Introduction

predicate
is a function with range 
\{ \text{TRUE}, \text{FALSE} \}
equivalence relation: a binary relation \( R \) satisfying
\begin{align*}
(1) & \quad \text{reflexive}: (x,x) \in R \\
(2) & \quad \text{symmetric}: \text{if } (x,y) \in R \text{ then } (y,x) \in R \\
(3) & \quad \text{transitive}: \text{if } (x,y) \in R \text{ and } (y,z) \in R, \text{ then } (x,z) \in R
\end{align*}
**predicate** is a function with range $= \{TRUE, FALSE\}$
Chapter 0 Introduction

**predicate** is a function with range $= \{TRUE, FALSE\}$

**equivalence relation**: a binary relation $R$ satisfying

1. **reflexive**: $(x,x) \in R$
2. **symmetric**: if $(x,y) \in R$ then $(y,x) \in R$
3. **transitive**: if $(x,y) \in R$ and $(y,z) \in R$, then $(x,z) \in R$
predicate is a function with range \( \{ TRUE, FALSE \} \)

equivalence relation: a binary relation \( R \) satisfying

(1) reflexive: \((x, x) \in R\)
Chapter 0 Introduction

**predicate** is a function with range \( \{ TRUE, FALSE \} \)

**equivalence relation**: a binary relation \( R \) satisfying

1. **reflexive**: \( (x, x) \in R \)
2. **symmetric**: if \( (x, y) \in R \) then \( (y, x) \in R \)
3. **transitive**: if \( (x, y) \in R \) and \( (y, z) \in R \), then \( (x, z) \in R \)
predicate is a function with range $= \{TRUE, FALSE\}$

equivalence relation: a binary relation $R$ satisfying

(1) reflexive: $(x, x) \in R$
(2) symmetric: if $(x, y) \in R$ then $(y, x) \in R$
(3) transitive: if $(x, y) \in R$ and $(y, z) \in R$, then $(x, z) \in R$
Chapter 0 Introduction

directed edge if \((u,v) \neq (v,u)\)

subgraph: \(H = (U,F)\) of \(G\), if \(U \subseteq V\) and \(F \subseteq E \cap (U \times U)\).

path: a sequence of vertices in \(V\)

simple path: the vertices on the path do not repeat

cycle: path in which the start and end vertices are the same

tree: graph without cycles

connected graph: there is a path between every two vertices
Chapter 0 Introduction

**Graph:** a pair of sets $(V, E)$, where $E$ is a relation $\subseteq V \times V$, elements $v \in V$ are vertices, elements $(u, v) \in E$ are edges.
Chapter 0 Introduction

**graph:** a pair of sets \((V, E)\), where \(E\) is a relation \(\subseteq V \times V\),
elements \(v \in V\) are vertices, elements \((u, v) \in E\) are edges.
directed edge if \((u, v) \neq (v, u)\)
Chapter 0 Introduction

graph: a pair of sets \((V, E)\), where \(E\) is a relation \(\subseteq V \times V\),
elements \(v \in V\) are vertices, elements \((u, v) \in E\) are edges.
directed edge if \((u, v) \neq (v, u)\)
subgraph: \(H = (U, F)\) of \(G\), if \(U \subseteq V\) and \(F \subseteq E \cap (U \times U)\).
Chapter 0 Introduction

**graph**: a pair of sets \((V, E)\), where \(E\) is a relation \(\subseteq V \times V\), elements \(v \in V\) are vertices, elements \((u, v) \in E\) are edges.

directed edge if \((u, v) \neq (v, u)\)

subgraph: \(H = (U, F)\) of \(G\), if \(U \subseteq V\) and \(F \subseteq E \cap (U \times U)\).

path: a sequence of vertices in \(V\)
graph: a pair of sets $(V, E)$, where $E$ is a relation $\subseteq V \times V$, elements $v \in V$ are vertices, elements $(u, v) \in E$ are edges.

directed edge if $(u, v) \neq (v, u)$

subgraph: $H = (U, F)$ of $G$, if $U \subseteq V$ and $F \subseteq E \cap (U \times U)$.

path: a sequence of vertices in $V$

simple path: the vertices on the path do not repeat
**Chapter 0 Introduction**

**graph**: a pair of sets $(V, E)$, where $E$ is a relation $\subseteq V \times V$, elements $v \in V$ are vertices, elements $(u, v) \in E$ are edges.

directed edge if $(u, v) \neq (v, u)$
subgraph: $H = (U, F)$ of $G$, if $U \subseteq V$ and $F \subseteq E \cap (U \times U)$.
path: a sequence of vertices in $V$
simple path: the vertices on the path do not repeat
cycle: path in which the start and end vertices are the same
graph: a pair of sets $(V, E)$, where $E$ is a relation $\subseteq V \times V$, elements $v \in V$ are vertices, elements $(u, v) \in E$ are edges.

directed edge if $(u, v) \neq (v, u)$
subgraph: $H = (U, F)$ of $G$, if $U \subseteq V$ and $F \subseteq E \cap (U \times U)$.
path: a sequence of vertices in $V$
simple path: the vertices on the path do not repeat
cycle: path in which the start and end vertices are the same
tree: graph without cycles
graph: a pair of sets \((V, E)\), where \(E\) is a relation \(\subseteq V \times V\), elements \(v \in V\) are vertices, elements \((u, v) \in E\) are edges.

directed edge if \((u, v) \neq (v, u)\)
subgraph: \(H = (U, F)\) of \(G\), if \(U \subseteq V\) and \(F \subseteq E \cap (U \times U)\).
path: a sequence of vertices in \(V\)
simple path: the vertices on the path do not repeat
cycle: path in which the start and end vertices are the same
tree: graph without cycles
connected graph: there is a path between every two vertices
Chapter 0 Introduction
Chapter 0 Introduction

String and Language
Chapter 0 Introduction

**String and Language**

alphabet: $\Sigma$, a finite set of symbols

string: $s$, a finite sequence of symbols taken from an alphabet

empty string: $\epsilon$ (without symbols)

set $\Sigma_0 = \{\epsilon\}$,

set $\Sigma_k = \{xy : x \in \Sigma_{k-1}, y \in \Sigma\}$, for $k = 1, 2, ...$

set $\Sigma^* = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup ...$

is called transitive closure of $\Sigma$

string is an element $\in \Sigma^*$

reverse of string $w$: $w^R$

string length: the number of symbols in a string

concatenation of strings $x = x_1 ... x_m$ and $y = y_1 ... y_n$:

$xy = x_1 ... x_m y_1 ... y_n$

lexicographical order of strings: dictionary order

language $L$: a set of strings, i.e., $L \subseteq \Sigma^*$
String and Language

alphabet: $\Sigma$, a finite set of symbols
string: $s$, a finite sequence of symbols taken from an alphabet
String and Language

alphabet: $\Sigma$, a finite set of symbols
string: $s$, a finite sequence of symbols taken from an alphabet
empty string: $\epsilon$ (without symbols)
String and Language

alphabet: $\Sigma$, a finite set of symbols

string: $s$, a finite sequence of symbols taken from an alphabet

empty string: $\epsilon$ (without symbols)

set $\Sigma^0 = \{\epsilon\}$,
String and Language

alphabet: $\Sigma$, a finite set of symbols
string: $s$, a finite sequence of symbols taken from an alphabet
empty string: $\epsilon$ (without symbols)
set $\Sigma^0 = \{\epsilon\}$,
set $\Sigma^k = \{xy : x \in \Sigma^{k-1}, y \in \Sigma\}$, for $k = 1, 2, \ldots$
String and Language

alphabet: $\Sigma$, a finite set of symbols
string: $s$, a finite sequence of symbols taken from an alphabet
empty string: $\epsilon$ (without symbols)
set $\Sigma^0 = \{\epsilon\}$,
set $\Sigma^k = \{xy : x \in \Sigma^{k-1}, y \in \Sigma\}$, for $k = 1, 2, \ldots$
set $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$
String and Language

alphabet: $\Sigma$, a finite set of symbols
string: $s$, a finite sequence of symbols taken from an alphabet
empty string: $\epsilon$ (without symbols)
set $\Sigma^0 = \{\epsilon\}$,
set $\Sigma^k = \{xy : x \in \Sigma^{k-1}, y \in \Sigma\}$, for $k = 1, 2, \ldots$
set $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$ is called transitive closure of $\Sigma$
String and Language

alphabet: $\Sigma$, a finite set of symbols
string: $s$, a finite sequence of symbols taken from an alphabet
empty string: $\epsilon$ (without symbols)
set $\Sigma^0 = \{\epsilon\}$,
set $\Sigma^k = \{xy : x \in \Sigma^{k-1}, y \in \Sigma\}$, for $k = 1, 2, \ldots$
set $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$ is called transitive closure of $\Sigma$
string is an element $\in \Sigma^*$
String and Language

alphabet: $\Sigma$, a finite set of symbols
string: $s$, a finite sequence of symbols taken from an alphabet
empty string: $\epsilon$ (without symbols)
set $\Sigma^0 = \{\epsilon\}$,
set $\Sigma^k = \{xy : x \in \Sigma^{k-1}, y \in \Sigma\}$, for $k = 1, 2, \ldots$
set $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$ is called transitive closure of $\Sigma$
string is an element $\in \Sigma^*$
reverse of string $w$: $w^R$
String and Language

alphabet: $\Sigma$, a finite set of symbols
string: $s$, a finite sequence of symbols taken from an alphabet
empty string: $\epsilon$ (without symbols)
set $\Sigma^0 = \{\epsilon\}$,
set $\Sigma^k = \{xy : x \in \Sigma^{k-1}, y \in \Sigma\}$, for $k = 1, 2, \ldots$
set $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$ is called transitive closure of $\Sigma$
string is an element $\in \Sigma^*$
reverse of string $w$: $w^R$
string length: the number of symbols in a string
Chapter 0 Introduction

String and Language

alphabet: \( \Sigma \), a finite set of symbols
string: \( s \), a finite sequence of symbols taken from an alphabet
empty string: \( \epsilon \) (without symbols)
set \( \Sigma^0 = \{\epsilon\} \),
set \( \Sigma^k = \{xy : x \in \Sigma^{k-1}, y \in \Sigma\} \), for \( k = 1, 2, \ldots \)
set \( \Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots \) is called transitive closure of \( \Sigma \)
string is an element \( \in \Sigma^* \)
reverse of string \( w \): \( w^R \)
string length: the number of symbols in a string
concatenation of strings \( x = x_1 \ldots x_m \) and \( y = y_1 \ldots y_n \):
\[ xy = x_1 \ldots x_m y_1 \ldots y_n \]
String and Language

alphabet: $\Sigma$, a finite set of symbols
string: $s$, a finite sequence of symbols taken from an alphabet
empty string: $\epsilon$ (without symbols)
set $\Sigma^0 = \{\epsilon\}$,
set $\Sigma^k = \{xy : x \in \Sigma^{k-1}, y \in \Sigma\}$, for $k = 1, 2, \ldots$
set $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$ is called transitive closure of $\Sigma$
string is an element $\in \Sigma^*$
reverse of string $w$: $w^R$
string length: the number of symbols in a string
concatenation of strings $x = x_1 \ldots x_m$ and $y = y_1 \ldots y_n$:
$$xy = x_1 \ldots x_m y_1 \ldots y_n$$
lexicographical order of strings: dictionary order
String and Language

alphabet: $\Sigma$, a finite set of symbols
string: $s$, a finite sequence of symbols taken from an alphabet
empty string: $\epsilon$ (without symbols)
set $\Sigma^0 = \{\epsilon\}$,
set $\Sigma^k = \{xy : x \in \Sigma^{k-1}, y \in \Sigma\}$, for $k = 1, 2, \ldots$
set $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$ is called transitive closure of $\Sigma$
string is an element $\in \Sigma^*$
reverse of string $w$: $w^R$
string length: the number of symbols in a string
concatenation of strings $x = x_1 \ldots x_m$ and $y = y_1 \ldots y_n$:
$$xy = x_1 \ldots x_my_1 \ldots y_n$$
lexicographical order of strings: dictionary order
language $L$: a set of strings, i.e., $L \subseteq \Sigma^*$
Chapter 0 Introduction

Boolean Logic:
boolean values: TRUE, FALSE, or 1, 0
boolean operands: ∧, ∨, ¬
boolean variables: x, y, z
boolean expressions: P, Q, R, formed by boolean values, operands, variables, and expressions.
Chapter 0 Introduction

Boolean Logic:

boolean values: TRUE, FALSE, or 1, 0

boolean operands: ∧, ∨, ¬

boolean variables: x, y, z

boolean expressions: P, Q, R, formed by boolean values, operands, variables, and expressions.
Chapter 0 Introduction

Boolean Logic:

boolean values: \textit{TRUE}, \textit{FALSE}, or 1, 0
Chapter 0 Introduction

Boolean Logic:

boolean values: $TRUE, FALSE$, or 1, 0
boolean operands: $∧, ∨, ¬$
Chapter 0 Introduction

**Boolean Logic:**

boolean values: \textit{TRUE}, \textit{FALSE}, or 1, 0  
boolean operands: \( \land, \lor, \neg \)  
boolean variables: \( x, y, z \)
Chapter 0 Introduction

Boolean Logic:

boolean values: \emph{TRUE}, \emph{FALSE}, or 1, 0
boolean operands: $\land$, $\lor$, $\neg$
boolean variables: $x$, $y$, $z$
boolean expressions: $P$, $Q$, $R$, formed by
  boolean values, operands, variables, and expressions.
Chapter 0 Introduction

Definition, theorem, and proof:
- definition: describing objects precisely
- mathematical statement: stating objects that have certain property.
- proof: a convincing logical argument that a statement is true
- theorem: a mathematical statement proved true
- lemma: a theorem assisting the proof of an more significant theorem
- corollary: conclusion easily derived from a theorem or lemma
Chapter 0 Introduction

Definition, theorem, and proof:
Chapter 0 Introduction

Definition, theorem, and proof:

definition: describing objects precisely
Definition, theorem, and proof:

definition: describing objects precisely
mathematical statement: stating objects that have certain property.
Chapter 0 Introduction

Definition, theorem, and proof:

definition: describing objects precisely
mathematical statement: stating objects that have certain property.
proof: a convincing logical argument that a statement is true
Chapter 0 Introduction

Definition, theorem, and proof:

definition: describing objects precisely
mathematical statement: stating objects that have certain property.
proof: a convincing logical argument that a statement is true
theorem: a mathematical statement proved true
Chapter 0 Introduction

Definition, theorem, and proof:

definition: describing objects precisely
mathematical statement: stating objects that have certain property.
proof: a convincing logical argument that a statement is true
theorem: a mathematical statement proved true
lemma: a theorem assisting the proof of an more significant theorem
Chapter 0 Introduction

Definition, theorem, and proof:

definition: describing objects precisely
mathematical statement: stating objects that have certain property.
proof: a convincing logical argument that a statement is true
theorem: a mathematical statement proved true
lemma: a theorem assisting the proof of an more significant theorem
corollary: conclusion easily derived from a theorem or lemma
Chapter 0 Introduction

Building proofs: not always easy, so be patient, be logical, be neat/concise.

Types of proofs:
- Proof by construction
- Proof by contradiction
- Proof by induction
Chapter 0 Introduction

Building proofs:

- Proof by construction
- Proof by contradiction
- Proof by induction
Building proofs:
not alway easy,
Chapter 0 Introduction

**Building proofs:**

not alway easy, so

be patient
be logical
be neat/concise
Building proofs:

not always easy, so

be patient
be logical
be neat/concise

type of proofs:
  proof by construction
  proof by contradiction
  proof by induction
Chapter 0 Introduction

Some theorems claim that some type of object (or property) exists, for example,
Theorem 1. $\forall n > 2$, there is a graph of $n$ vertices in which every vertex has degree 3.

Proof: Construct such a graph for every given $n > 2$ even. Construct a "cycle" using all $n$ vertices, and create edges $(i, i + n/2)$ for all $i = 1, 2, ..., n/2$. 


proof by construction

Chapter 0 Introduction

Some theorems claim that some type of object (or property) exists, for example,

\[ \forall n > 2, \text{there is a graph of } n \text{ vertices in which every vertex has degree 3.} \]

Proof: Construct such a graph for every given \( n > 2 \) even. Construct a "cycle" using all \( n \) vertices, and create edges \((i, i + n/2)\) for all \( i = 1, 2, \ldots, n/2 \).
proof by construction

Some theorems claim that some type of object (or property) exists,
proof by construction

Some theorems claim that some type of object (or property) exists, for example,

Theorem 1. \( \forall n > 2 \), there is a graph of \( n \) vertices in which every vertex has degree 3.
proof by construction

Some theorems claim that some type of object (or property) exists, for example,

Theorem 1. $\forall n > 2$, there is a graph of $n$ vertices in which every vertex has degree 3.

Proof: Construct such a graph for every given $n > 2$ even.
proof by construction

Some theorems claim that some type of object (or property) exists, for example,

Theorem 1. $\forall n > 2$, there is a graph of $n$ vertices in which every vertex has degree 3.

**Proof:** Construct such a graph for every given $n > 2$ even.

Construct a ”cycle” using all $n$ vertices, and create edges $(i, i + n/2)$ for all $i = 1, 2, \ldots, n/2$. 
Chapter 0 Introduction

Theorem 2 (Pigeonhole principle). Placing $n$ pigeons in $k$ holes, $k < n$, there is at least one hole hosting more than one pigeon.

Proof: Assume otherwise; then the total number of pigeons is $\leq k < n$. Contradicts.
proof by contradiction
proof by contradiction

Theorem 2 (*Pigeonhole principle*). Placing $n$ pigeons in $k$ holes, $k < n$, there is at least one hole hosting more than one pigeon.
proof by contradiction

Theorem 2 (Pigeonhole principle). Placing $n$ pigeons in $k$ holes, $k < n$, there is at least one hole hosting more than one pigeon.

Proof: Assume otherwise;
proof by contradiction

Theorem 2 (*Pigeonhole principle*). Placing \( n \) pigeons in \( k \) holes, \( k < n \), there is at least one hole hosting more than one pigeon.

**Proof:** Assume otherwise;

Then the total number of pigeons is \( \leq k \)
proof by contradiction

Theorem 2 (Pigeonhole principle). Placing \( n \) pigeons in \( k \) holes, \( k < n \), there is at least one hole hosting more than one pigeon.

Proof: Assume otherwise;

Then the total number of pigeons is \( \leq k < n \). Contradicts.
Chapter 0 Introduction

Theorem 3. $\sqrt{2}$ is irrational.

(A number is rational if it can be expressed as $m/n$ for two integers $m$ and $n$.)

Proof: Assume otherwise, i.e., $\sqrt{2} = m/n$ for some $m$ and $n$ (at least one of $m$ and $n$ is odd!).

Then $n\sqrt{2} = m$; $2n^2 = m^2$; $m^2$ is even so $m$ cannot be odd (why?).

$m$ can be written as $m = 2k$, for some integer $k$.

And $2n^2 = m^2 = 4k^2 = \Rightarrow n^2 = 2k^2$.

So $n^2$ is even; thus $n$ cannot be odd.

contradicts that at least one of $m$ and $n$ is odd!

the assumption $\sqrt{2} = m/n$ was wrong.
Theorem 3. $\sqrt{2}$ is irrational.

(A number is rational if it can be expressed as $m/n$ for two integers $m$ and $n$.)
Chapter 0 Introduction

Theorem 3. $\sqrt{2}$ is irrational.

(A number is rational if it can be expressed as $m/n$ for two integers $m$ and $n$.)

Proof: Assume otherwise,
Chapter 0 Introduction

Theorem 3. $\sqrt{2}$ is irrational.

(A number is rational if it can be expressed as $m/n$ for two integers $m$ and $n$.)

Proof: Assume otherwise, i.e., $\sqrt{2} = m/n$ for some $m$ and $n$ (at least one of $m$ and $n$ is odd!).
Theorem 3. $\sqrt{2}$ is irrational.

(A number is rational if it can be expressed as $m/n$ for two integers $m$ and $n$.)

Proof: Assume otherwise, i.e., $\sqrt{2} = m/n$ for some $m$ and $n$ 
(at least one of $m$ and $n$ is odd!).

Then $n\sqrt{2} = m$;
Chapter 0 Introduction

Theorem 3. \( \sqrt{2} \) is irrational.

(A number is rational if it can be expressed as \( m/n \) for two integers \( m \) and \( n \).)

Proof: Assume otherwise, i.e., \( \sqrt{2} = m/n \) for some \( m \) and \( n \) (at least one of \( m \) and \( n \) is odd!).

Then \( n\sqrt{2} = m \);

\( 2n^2 = m^2 \).
Chapter 0 Introduction

Theorem 3. $\sqrt{2}$ is irrational.

(A number is rational if it can be expressed as $m/n$ for two integers $m$ and $n$.)

Proof: Assume otherwise, i.e., $\sqrt{2} = m/n$ for some $m$ and $n$

(at least one of $m$ and $n$ is odd!).

Then $n\sqrt{2} = m$;

$2n^2 = m^2$; $m^2$ is even
so $m$ cannot be odd (why?)
Chapter 0 Introduction

Theorem 3. $\sqrt{2}$ is irrational.

(A number is rational if it can be expressed as $m/n$ for two integers $m$ and $n$.)

Proof: Assume otherwise, i.e., $\sqrt{2} = m/n$ for some $m$ and $n$ 
(at least one of $m$ and $n$ is odd!).

Then $n\sqrt{2} = m$;

$2n^2 = m^2$; $m^2$ is even
so $m$ cannot be odd (why?)

$m$ can be written as $m = 2k$, for some integer $k$. 

Chapter 0 Introduction

Theorem 3. $\sqrt{2}$ is irrational.

(A number is rational if it can be expressed as $m/n$ for two integers $m$ and $n$.)

**Proof:** Assume otherwise, i.e., $\sqrt{2} = m/n$ for some $m$ and $n$ (at least one of $m$ and $n$ is odd!).

Then $n\sqrt{2} = m$;

$2n^2 = m^2$; $m^2$ is even
so $m$ cannot be odd (why?)

$m$ can be written as $m = 2k$, for some integer $k$. And

$2n^2 = m^2$
Chapter 0 Introduction

Theorem 3. √2 is irrational.

(A number is rational if it can be expressed as m/n for two integers m and n.)

Proof: Assume otherwise, i.e., √2 = m/n for some m and n
(at least one of m and n is odd!).

Then \( n\sqrt{2} = m \);

2\(n^2 = m^2 \); \(m^2 \) is even
so m cannot be odd (why ?)

\(m \) can be written as \(m = 2k \), for some integer \(k\). And

\[ 2n^2 = m^2 \implies 2n^2 = 4k^2 \]


Chapter 0 Introduction

Theorem 3. $\sqrt{2}$ is irrational.

(A number is rational if it can be expressed as $m/n$ for two integers $m$ and $n$.)

Proof: Assume otherwise, i.e., $\sqrt{2} = m/n$ for some $m$ and $n$ (at least one of $m$ and $n$ is odd!).

Then $n\sqrt{2} = m$;

$2n^2 = m^2$; $m^2$ is even
so $m$ cannot be odd (why?)

$m$ can be written as $m = 2k$, for some integer $k$. And

$$2n^2 = m^2 \implies 2n^2 = 4k^2 \implies n^2 = 2k^2$$
Chapter 0 Introduction

Theorem 3. $\sqrt{2}$ is irrational.
(A number is rational if it can be expressed as $m/n$ for two integers $m$ and $n$.)

**Proof:** Assume otherwise, i.e., $\sqrt{2} = m/n$ for some $m$ and $n$ 
(at least one of $m$ and $n$ is odd!).

Then $n\sqrt{2} = m$;

$2n^2 = m^2$; $m^2$ is even
so $m$ cannot be odd (why?)

$m$ can be written as $m = 2k$, for some integer $k$. And

$$2n^2 = m^2 \implies 2n^2 = 4k^2 \implies n^2 = 2k^2$$

So $n^2$ is even; thus $n$ cannot be odd.
Chapter 0 Introduction

Theorem 3. $\sqrt{2}$ is irrational.

(A number is rational if it can be expressed as $m/n$ for two integers $m$ and $n$.)

**Proof:** Assume otherwise, i.e., $\sqrt{2} = m/n$ for some $m$ and $n$ (at least one of $m$ and $n$ is odd!).

Then $n\sqrt{2} = m$;

$2n^2 = m^2$; $m^2$ is even
so $m$ cannot be odd (why?)

$m$ can be written as $m = 2k$, for some integer $k$. And

$$2n^2 = m^2 \implies 2n^2 = 4k^2 \implies n^2 = 2k^2$$

So $n^2$ is even; thus $n$ cannot be odd.

contradicts that at least one of $m$ and $n$ is odd!

the assumption $\sqrt{2} = m/n$ was wrong.
Chapter 0 Introduction

To show certain property $P(n)$ holds for every integer $n = 1, 2, ...$, it suffices to show

1. $P(1)$, the property holds for $n = 1$,
2. if the property holds for $k$, then it holds for $k + 1$, i.e., $P(k) \rightarrow P(k + 1)$

where (2) is "chain reaction" or "property propagation", while (1) is the "starting point".

like knocking dominos: all dominos will be down because

1. The first domino will be pushed down, and
2. if any domino is knocked down, the one behind it will be down also.
proof by induction

To show certain property $P(n)$ holds for every integer $n = 1, 2, ...$, it suffices to show (1) $P(1)$, the property holds for $n = 1$, (2) if the property holds for $k$, then it holds for $k + 1$, i.e., $P(k) \rightarrow P(k + 1)$ where (2) is "chain reaction" or "property propagation", while (1) is the "starting point". Like knocking dominos: all dominos will be down because (1) The first domino will be pushed down, and (2) if any domino is knocked down, the one behind it will be down also.
proof by induction

To show certain property $P(n)$ holds for every integer $n = 1, 2, \ldots$, it suffices to show

(1) $P(1)$, the property holds for $n = 1$,
(2) if the property holds for $k$, then it holds for $k + 1$, i.e., $P(k) \rightarrow P(k + 1)$

where (2) is "chain reaction" or "property propagation", while (1) is the "starting point".

like knocking dominos: all dominos will be down because

(1) The first domino will be pushed down, and
(2) if any domino is knocked down, the one behind it will be down also.
proof by induction

To show certain property \( P(n) \) holds for every integer \( n = 1, 2, \ldots \), it suffices to show

1. \( P(1) \), the property holds for \( n = 1 \),
proof by induction

To show certain property $P(n)$ holds for every integer $n = 1, 2, \ldots$, It suffices to show

(1) $P(1)$, the property holds for $n = 1$,
(2) if the property holds for $k$, then it holds for $k + 1$, 
proof by induction

To show certain property $\mathcal{P}(n)$ holds for every integer $n = 1, 2, \ldots$, it suffices to show

(1) $\mathcal{P}(1)$, the property holds for $n = 1$,
(2) if the property holds for $k$, then it holds for $k + 1$, i.e.,

$$\mathcal{P}(k) \implies \mathcal{P}(k + 1)$$
proof by induction

To show certain property $\mathcal{P}(n)$ holds for every integer $n = 1, 2, \ldots$, it suffices to show

1. $\mathcal{P}(1)$, the property holds for $n = 1$,
2. if the property holds for $k$, then it holds for $k + 1$, i.e.,

$$\mathcal{P}(k) \rightarrow \mathcal{P}(k + 1)$$

where (2) is “chain reaction” or “property propagation”,

like knocking dominos: all dominos will be down because

1. The first domino will be pushed down, and
2. if any domino is knocked down, the one behind it will be down also.
proof by induction

To show certain property $\mathcal{P}(n)$ holds for every integer $n = 1, 2, \ldots$, it suffices to show

(1) $\mathcal{P}(1)$, the property holds for $n = 1$,
(2) if the property holds for $k$, then it holds for $k + 1$, i.e.,

$$\mathcal{P}(k) \rightarrow \mathcal{P}(k + 1)$$

where (2) is “chain reaction” or “property propagation”, while (1) is the ”starting point”.
proof by induction

To show certain property $\mathcal{P}(n)$ holds for every integer $n = 1, 2, \ldots$, it suffices to show

1. $\mathcal{P}(1)$, the property holds for $n = 1$,
2. if the property holds for $k$, then it holds for $k + 1$, i.e.,

$$\mathcal{P}(k) \rightarrow \mathcal{P}(k + 1)$$

where (2) is “chain reaction” or “property propagation”, while (1) is the ”starting point”.

like knocking dominos: all dominos will be down because
proof by induction

To show certain property $\mathcal{P}(n)$ holds for every integer $n = 1, 2, \ldots$, it suffices to show

(1) $\mathcal{P}(1)$, the property holds for $n = 1$,
(2) if the property holds for $k$, then it holds for $k + 1$, i.e.,

$$\mathcal{P}(k) \rightarrow \mathcal{P}(k + 1)$$

where (2) is "chain reaction" or "property propagation", while (1) is the "starting point".

like knocking dominos: all dominos will be down because

(1) The first domino will be pushed down, and
proof by induction

To show certain property $\mathcal{P}(n)$ holds for every integer $n = 1, 2, \ldots$, it suffices to show

(1) $\mathcal{P}(1)$, the property holds for $n = 1$,
(2) if the property holds for $k$, then it holds for $k + 1$, i.e.,

$$\mathcal{P}(k) \rightarrow \mathcal{P}(k + 1)$$

where (2) is “chain reaction” or “property propagation”, while (1) is the ”starting point”.

like knocking dominos: all dominos will be down because

(1) The first domino will be pushed down, and
(2) if any domino is knocked down, the one behind it will be down also.
Chapter 0 Introduction

Theorem 4. For all \( n \geq 1 \), summation 
\[
1 + 2 + \cdots + n = n^2 \left( n + 1 \right)
\]

We use proof by induction to prove. What is \( P \) here?

\[
P(n) \equiv 1 + 2 + \cdots + n = n^2 \left( n + 1 \right)
\]

Proof. \( n = 1 \), \( P(n) \) holds because 
\[
1 = 1^2 \left( 1 + 1 \right)
\]

Assume \( P(k) \) holds, i.e., 
\[
1 + 2 + \cdots + k = k^2 \left( k + 1 \right)
\]

We now show \( P(k+1) \) holds as well
\[
1 + 2 + \cdots + k + k + 1 = \left(1 + 2 + \cdots + k\right) + (k + 1)
= k^2 \left( k + 1 \right) + (k + 1)
= (k + 1) \left( k/2 + 1 \right)
= (k + 1) \left( k/2 + 2/2 \right)
= (k + 1)^2 \left( k + 2 \right)
\]

We have proved \( P(n) \) holds for all \( n \geq 1 \).
Theorem 4. For all $n \geq 1$, summation $1 + 2 + \cdots + n = \frac{n}{2}(n + 1)$
Chapter 0 Introduction

Theorem 4. For all $n \geq 1$, summation $1 + 2 + \cdots + n = \frac{n}{2}(n + 1)$
We use proof by induction to prove.
Theorem 4. For all $n \geq 1$, summation $1 + 2 + \cdots + n = \frac{n}{2}(n + 1)$
We use proof by induction to prove.

What is $\mathcal{P}$ here?
Theorem 4. For all \( n \geq 1 \), summation \( 1 + 2 + \cdots + n = \frac{n}{2}(n + 1) \)
We use proof by induction to prove.

What is \( P \) here?

\[
P(n) \equiv "1 + 2 + \cdots + n = \frac{n}{2}(n + 1)"
\]
Theorem 4. For all \( n \geq 1 \), summation \( 1 + 2 + \cdots + n = \frac{n}{2}(n + 1) \)
We use proof by induction to prove.

What is \( P \) here?

\[ P(n) \equiv "1 + 2 + \cdots + n = \frac{n}{2}(n + 1)" \]

Proof.
Chapter 0 Introduction

Theorem 4. For all \( n \geq 1 \), summation \( 1 + 2 + \cdots + n = \frac{n}{2}(n + 1) \)
We use proof by induction to prove.

What is \( \mathcal{P} \) here?

\[ \mathcal{P}(n) \equiv \text{“}1 + 2 + \cdots + n = \frac{n}{2}(n + 1)\text{”} \]

Proof. \( n = 1 \), \( \mathcal{P}(n) \) holds because \( 1 = \frac{1}{2}(1 + 1) \).
Chapter 0 Introduction

Theorem 4. For all $n \geq 1$, summation $1 + 2 + \cdots + n = \frac{n}{2}(n + 1)$
We use proof by induction to prove.

What is $\mathcal{P}$ here?

$\mathcal{P}(n) \equiv \text{“}1 + 2 + \cdots + n = \frac{n}{2}(n + 1)\text{”}$

Proof. $n = 1$, $\mathcal{P}(n)$ holds because $1 = \frac{1}{2}(1 + 1)$.

Assume $\mathcal{P}(k)$ holds, i.e., $1 + 2 + \cdots + k = \frac{k}{2}(k + 1)$
Chapter 0 Introduction

Theorem 4. For all \( n \geq 1 \), summation \( 1 + 2 + \cdots + n = \frac{n}{2}(n + 1) \)
We use proof by induction to prove.

What is \( \mathcal{P} \) here?

\[ \mathcal{P}(n) \equiv \text{“} 1 + 2 + \cdots + n = \frac{n}{2}(n + 1) \text{”} \]

Proof. \( n = 1 \), \( \mathcal{P}(n) \) holds because \( 1 = \frac{1}{2}(1 + 1) \).

Assume \( \mathcal{P}(k) \) holds, i.e., \( 1 + 2 + \cdots + k = \frac{k}{2}(k + 1) \)

We now show \( \mathcal{P}(k + 1) \) holds as well
Chapter 0 Introduction

Theorem 4. For all $n \geq 1$, summation $1 + 2 + \cdots + n = \frac{n}{2}(n + 1)$

We use proof by induction to prove.

What is $P$ here?

$P(n) \equiv "1 + 2 + \cdots + n = \frac{n}{2}(n + 1)"

Proof. $n = 1$, $P(n)$ holds because $1 = \frac{1}{2}(1 + 1)$.

Assume $P(k)$ holds, i.e., $1 + 2 + \cdots + k = \frac{k}{2}(k + 1)$

We now show $P(k + 1)$ holds as well

$1 + 2 + \cdots + k + k + 1 = (1 + 2 + \cdots + k) + (k + 1)$
Chapter 0 Introduction

Theorem 4. For all $n \geq 1$, summation $1 + 2 + \cdots + n = \frac{n}{2}(n + 1)$
We use proof by induction to prove.

What is $P$ here?

$P(n) \equiv "1 + 2 + \cdots + n = \frac{n}{2}(n + 1)"

Proof. $n = 1$, $P(n)$ holds because $1 = \frac{1}{2}(1 + 1)$.

Assume $P(k)$ holds, i.e., $1 + 2 + \cdots + k = \frac{k}{2}(k + 1)$

We now show $P(k + 1)$ holds as well

$1 + 2 + \cdots + k + k + 1 = (1 + 2 + \cdots + k) + (k + 1)$

$= \frac{k}{2}(k + 1) + (k + 1)$
Chapter 0 Introduction

Theorem 4. For all $n \geq 1$, summation $1 + 2 + \cdots + n = \frac{n}{2} (n + 1)$
We use proof by induction to prove.

What is $P$ here?

$$P(n) \equiv "1 + 2 + \cdots + n = \frac{n}{2} (n + 1)"$$

Proof. $n = 1$, $P(n)$ holds because $1 = \frac{1}{2} (1 + 1)$.

Assume $P(k)$ holds, i.e., $1 + 2 + \cdots + k = \frac{k}{2} (k + 1)$

We now show $P(k + 1)$ holds as well

$$1 + 2 + \cdots + k + k + 1 = (1 + 2 + \cdots + k) + (k + 1)$$
$$= \frac{k}{2} (k + 1) + (k + 1)$$
$$= (k + 1)(k/2 + 1)$$
Chapter 0 Introduction

Theorem 4. For all \( n \geq 1 \), summation \( 1 + 2 + \cdots + n = \frac{n}{2}(n + 1) \)
We use proof by induction to prove.

What is \( \mathcal{P} \) here?

\[ \mathcal{P}(n) \equiv \text{“}1 + 2 + \cdots + n = \frac{n}{2}(n + 1)\text{”} \]

Proof. \( n = 1 \), \( \mathcal{P}(n) \) holds because \( 1 = \frac{1}{2}(1 + 1) \).

Assume \( \mathcal{P}(k) \) holds, i.e., \( 1 + 2 + \cdots + k = \frac{k}{2}(k + 1) \)

We now show \( \mathcal{P}(k+1) \) holds as well
\[
1 + 2 + \cdots + k + k + 1 = (1 + 2 + \cdots + k) + (k + 1) \\
= \frac{k}{2}(k + 1) + (k + 1) \\
= (k + 1)(k/2 + 1) \\
= (k + 1)(k/2 + 2/2)
\]
Theorem 4. For all $n \geq 1$, summation $1 + 2 + \cdots + n = \frac{n}{2}(n + 1)$

We use proof by induction to prove.

What is $P$ here?

$P(n) \equiv "1 + 2 + \cdots + n = \frac{n}{2}(n + 1)"

Proof. $n = 1$, $P(n)$ holds because $1 = \frac{1}{2}(1 + 1)$.

Assume $P(k)$ holds, i.e., $1 + 2 + \cdots + k = \frac{k}{2}(k + 1)$

We now show $P(k + 1)$ holds as well

$1 + 2 + \cdots + k + k + 1 = (1 + 2 + \cdots + k) + (k + 1)$

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

$= (k + 1)(k/2 + 1)$

$= (k + 1)(k/2 + 2/2)$

$= \frac{k+1}{2}(((k + 1) + 1)$
Chapter 0 Introduction

Theorem 4. For all \( n \geq 1 \), summation \( 1 + 2 + \cdots + n = \frac{n}{2}(n + 1) \)

We use proof by induction to prove.

What is \( P \) here?

\[ P(n) \equiv "1 + 2 + \cdots + n = \frac{n}{2}(n + 1)" \]

Proof. \( n = 1 \), \( P(n) \) holds because \( 1 = \frac{1}{2}(1 + 1) \).

Assume \( P(k) \) holds, i.e., \( 1 + 2 + \cdots + k = \frac{k}{2}(k + 1) \)

We now show \( P(k + 1) \) holds as well

\[
1 + 2 + \cdots + k + k + 1 = (1 + 2 + \cdots + k) + (k + 1) \\
= \frac{k}{2}(k + 1) + (k + 1) \\
= (k + 1)(k/2 + 1) \\
= (k + 1)(k/2 + 2/2) \\
= \frac{k+1}{2}((k + 1) + 1)
\]

We have proved \( P(n) \) holds for all \( n \geq 1 \).