CSCI 2670 Introduction to Theory of Computing

Lecture Note 1
Introduction and Review

Liming Cai
CS@UGA, Fall 2018
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)
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
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-completeness
   - Chapter 9: intractability

The final 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
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
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

The final exam
Motivations

Why the theory course?

- Are basic programming skills (e.g., learnt from 1301/1302) sufficient for your career challenges?
- CS discipline should make you at least a savvy software engineer
Motivations

1. The 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. An elegant, simple way to think about computation: fundamental issues remain regardless of advancements of technologies.
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 advancements of technologies.
Motivations

Motivations:

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

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 advancements of technologies
Motivations

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 advancements of technologies

4. emerging applications
   internet search, bio-medical sciences
Motivations

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 advancements of technologies

4. emerging applications
   internet search, bio-medical sciences

5. non-traditional computation models
   quantum computers, bio/chem inspired computers
Objectives

What are the goals of this course?

Summarized as the Chomsky Hierarchy and extension Model Languages/Sets What are they?

1. finite automata
   - regular constant memory
2. push-down FA
   - context-free with an additional stack
3. linear Turing machines
   - context-sensitive memory proportional to input length
4. Turing machines
   - decidable unlimited memory
5. unknown
   - undecidable not computable by TMs

1. Polynomial-time TMs class P efficiently computable problems
2. Polynomial-time NTMs class NP intractable (but computable)
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/Sets</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>finite automata</td>
<td>regular constant memory</td>
<td></td>
</tr>
<tr>
<td>push-down FA</td>
<td>context-free with an additional stack</td>
<td></td>
</tr>
<tr>
<td>linear Turing machines</td>
<td>context-sensitive memory proportional to input length</td>
<td></td>
</tr>
<tr>
<td>Turing machines</td>
<td>decidable unlimited memory</td>
<td></td>
</tr>
<tr>
<td>unknown</td>
<td>undecidable not computable by TMs</td>
<td></td>
</tr>
<tr>
<td>Polynomial-time TMs class P</td>
<td>efficiently computable problems</td>
<td></td>
</tr>
<tr>
<td>Polynomial-time NTMs</td>
<td>class NP intractable (but computable)</td>
<td></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/Sets</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>memory proportional to input length</td>
</tr>
<tr>
<td>linear Turing machines</td>
<td>context-sensitive</td>
<td>memory proportional to input length</td>
</tr>
<tr>
<td>Turing machines</td>
<td>decidable</td>
<td>unlimited memory</td>
</tr>
<tr>
<td>unknown</td>
<td>undecidable</td>
<td>not computable by TMs</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/Sets</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>with 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/Sets</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. finite automata</td>
<td>regular</td>
<td>constant memory</td>
</tr>
<tr>
<td>2. push-down FA</td>
<td>context-free</td>
<td>with an additional stack</td>
</tr>
<tr>
<td>3. linear Turing machines</td>
<td>context-sensitive</td>
<td>memory proportional to input length</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/Sets</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. finite automata</td>
<td>regular</td>
<td>constant memory</td>
</tr>
<tr>
<td>2. push-down FA</td>
<td>context-free</td>
<td>with an additional stack</td>
</tr>
<tr>
<td>3. linear Turing machines</td>
<td>context-sensitive</td>
<td>memory proportional to input length</td>
</tr>
<tr>
<td>4. Turing machines</td>
<td>decidable</td>
<td>unlimited memory</td>
</tr>
<tr>
<td>5. unknown</td>
<td></td>
<td>undecidable not computable by TMs</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/Sets</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. finite automata</td>
<td>regular</td>
<td>constant memory</td>
</tr>
<tr>
<td>2. push-down FA</td>
<td>context-free</td>
<td>with an additional stack</td>
</tr>
<tr>
<td>3. linear Turing machines</td>
<td>context-sensitive</td>
<td>memory proportional to input length</td>
</tr>
<tr>
<td>4. Turing machines</td>
<td>decidable</td>
<td>unlimited memory</td>
</tr>
<tr>
<td>5. unknown</td>
<td>undecidable</td>
<td>not computable by TMs</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/Sets</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. finite automata</td>
<td>regular</td>
<td>constant memory</td>
</tr>
<tr>
<td>2. push-down FA</td>
<td>context-free</td>
<td>with an additional stack</td>
</tr>
<tr>
<td>3. linear Turing machines</td>
<td>context-sensitive</td>
<td>memory proportional to input length</td>
</tr>
<tr>
<td>4. Turing machines</td>
<td>decidable</td>
<td>unlimited memory</td>
</tr>
<tr>
<td>5. unknown</td>
<td>undecidable</td>
<td>not computable by TMs</td>
</tr>
</tbody>
</table>

| 1. Polynomial-time TMs      | class P          | efficiently computable problems                            |
## Objectives

What are the goals of this course?

Summarized as the **Chomsky Hierarchy** and extension

<table>
<thead>
<tr>
<th>Model</th>
<th>Languages/Sets</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. finite automata</td>
<td>regular</td>
<td>constant memory with an additional stack</td>
</tr>
<tr>
<td>2. push-down FA</td>
<td>context-free</td>
<td>memory proportional to input length</td>
</tr>
<tr>
<td>3. linear Turing machines</td>
<td>context-sensitive</td>
<td>unlimited memory</td>
</tr>
<tr>
<td>4. Turing machines</td>
<td>decidable</td>
<td>not computable by TMs</td>
</tr>
<tr>
<td>5. unknown</td>
<td>undecidable</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Languages/Sets</th>
<th>What are they?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Polynomial-time TMs</td>
<td>class P</td>
<td>efficiently computable problems</td>
</tr>
<tr>
<td>2. Polynomial-time NTMs</td>
<td>class NP</td>
<td>intractable (but computable)</td>
</tr>
</tbody>
</table>
Can machines be more intelligent than human?

Theoretically, all functions computed by machine learning algorithms fall are computable by Turing machines; practically, not all functions computable by Turing machines can be described by human beings; but ML can approximate such functions and those beyond Turing machine-computable if there are sufficient data.
Can machines be more intelligent than human?

- **Theoretically**, all functions computed by machine learning algorithms fall are computable by Turing machines;

- **Practically**, not all functions computable by Turing machines can be described by human beings;

  but ML can approximate such functions and those beyond Turing machine-computable
Can machines be more intelligent than human?

- **Theoretically**, all functions computed by machine learning algorithms fall are computable by Turing machines;

- **Practically**, **not all functions** computable by Turing machines can be **described** by human beings;

**but ML can approximate** such functions and those beyond Turing machine-computable **if there are sufficient data**:
Some explanations for Chomsky Hierarchy

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

What can a program do if using only 4 variables

\[ \text{Int } X, Y, Z, W; \]

\[ \text{can only memorize small amount of information} \]

\[ \text{what can it not do} \]

\[ \text{cannot count} \]

\[ \text{e.g., cannot correctly recognize long expressions like } (x + 20) \times ((y - z) \times (w + u) - 40) \times v \]

\[ \text{Such a problem is “context-free”, while the program with limited memory is just “regular”} \]
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

```c
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?

Cannot count!

E.g., cannot correctly recognize long expressions like

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

Or simply

    ( ) ( ( ( ) ) )

Such a problem is "context-free",
while the program with limited memory is just "regular".
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

```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

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

Such a problem is "context-free", while the program with limited memory is just "regular".
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

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

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;
```

It 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

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

Such a problem is "context-free", 
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

```
()(()(()(())))
```

Such a problem is "context-free", while the program with limited memory is just "regular".
Some explanations for Chomsky Hierarchy

A stack would help!

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

Stack has unlimited memory but access is in a very restricted way.

E.g., A stack can help recognize the set of strings like

aa...abb...b

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

Also recognizing palindrom strings like

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!

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

Stack has unlimited memory but access is in a very restricted way.

E.g., A stack can help recognize the set of strings like
aa...abb...b
(with the same number of a’s and b’s)

Also recognizing palindroms strings like
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
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 very restricted way.

E.g., A stack can help recognize the set of strings like

```
```

aa...abb...b

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

Also recognizing palindroms strings like

```
```

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?

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

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

E.g., A stack can help recognize the set of strings like \texttt{aa\ldots ab\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 very restricted way.

E.g., A stack can help recognize the set of strings like \textit{aa\ldots abb\ldots b}
(with the same number of a’s and b’s)

Also recognizing palindroms strings like \textit{xy\ldots zz\ldots 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 very restricted way.

E.g., A stack can help recognize the set of strings like $aa\ldots abb\ldots b$
(with the same number of a’s and b’s)

Also recognizing palindroms strings like $xy\ldots zz\ldots yx$

But can strings $aa\ldots abb\ldots bcc\ldots c$ 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 very restricted way.

E.g., A stack can help recognize the set of strings like \textit{aa...abb...b} (with the same number of a’s and b’s)

Also recognizing palindroms strings like \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.
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 very restricted way.

E.g., A stack can help recognize the set of strings like $aa\ldots abb\ldots b$
(with the same number of a’s and b’s)

Also recognizing palindroms strings like $xy\ldots zz\ldots yx$

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 

\[ \text{aa...abb...b} \]

and

\[ \text{aa...a bb...b cc...c} \]

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

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

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

What is the fundamental difference between strings

\texttt{aa...abb...b} and

\texttt{aa...a}

\texttt{bb...b}

\texttt{cc...c}

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

\texttt{((()))}

\texttt{and}

\texttt{((()))(()})

and such

\texttt{123456 12345678910}

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

But

\texttt{((()))}}

\texttt{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

\[ aa\ldots abb\ldots b \] and

\[ aa\ldots abb...bcc\ldots c \]?
Some explanations for Chomsky Hierarchy

What is the fundamental difference between strings

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

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

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

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

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

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

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

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

\texttt{aa...abb...b and aa...abb...b c c...c}

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

\texttt{((())) and (((())))) and such}

\begin{align*}
&123456 & 12345678910 \\
&3-4, 2-5, 1-6, 8-9, \text{ and } 7-10.
\end{align*}

\texttt{But (((()))]]]]}

\begin{align*}
&123456789 \\
&\text{would need pairings } 3-4, 2-5, 1-6, 6-7, 5-8. \text{ and } 4-9
\end{align*}
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,

1. non-nested or non-crossing correlation patterns, easy problem: can be handled with limited (finite) memory

2. nested correlation pattern, moderately hard problems, can be handled with a stack (infinite but restricted-access memory)

3. crossing patterns, hard problems, need to be handled with 2 stacks (infinite, random access memory)

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

Intuitively,

1. non-nested or non-crossing **correlation** patterns,
Some explanations for Chomsky Hierarchy

Intuitively,

1. non-nested or non-crossing correlation patterns,
   easy problem: can be handled with limited (finite) memory
Some explanations for Chomsky Hierarchy

Intuitively,

1. non-nested or non-crossing correlation patterns,
   easy problem: can be handled with limited (finite) memory

2. nested correlation pattern,
Some explanations for Chomsky Hierarchy

Intuitively,

1. non-nested or non-crossing correlation patterns, 
   easy problem: can be handled with limited (finite) memory

2. nested correlation pattern, moderately hard problems, 
   can be handled with a stack (infinite but restricted-access memory)
Some explanations for Chomsky Hierarchy

Intuitively,

1. non-nested or non-crossing correlation patterns,
   easy problem: can be handled with limited (finite) memory

2. nested correlation pattern, moderately hard problems,
   can be handled with a stack (infinite but restricted-access memory)

3. crossing patterns,
Some explanations for Chomsky Hierarchy

Intuitively,

1. non-nested or non-crossing correlation patterns,
   easy problem: can be handled with limited (finite) memory

2. nested correlation pattern, moderately hard problems,
   can be handled with a stack (infinite but restricted-access memory)

3. crossing patterns, hard problems,
   need to be handled with 2 stacks (infinite, random access memory)
Some explanations for Chomsky Hierarchy

Intuitively,

1. non-nested or non-crossing correlation patterns, easy problem: can be handled with limited (finite) memory

2. nested correlation pattern, moderately hard problems, can be handled with a stack (infinite but restricted-access memory)

3. crossing patterns, hard problems, need to be handled with 2 stacks (infinite, random access memory)

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

Review of mathematical concepts

• sets, basic set operations, properties
• relations, functions, predicates
• strings, languages,
• Boolean logic, theorem, proofs
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

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): $A \times B = \{ (a,b) : a \in A, b \in B \}$
- Power set: $P(A) = \{ B : B \subseteq A \}$, or also denoted with $2^A$
**Set**: a collection of (related, discrete) objects
Chapter 0 Introduction

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: \( \phi \)
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
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 \( A \subset B \)
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: $\overline{S}$
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: $\overline{S}$
- 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):

\[
A \times B = \{(a, b) : a \in A, b \in B\}
\]
**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: $\overline{S}$
- 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. however, 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

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


**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** \( f : X \to Y \), 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 \( R_f = \{(1, -3), (2, 2), (3, 4)\} \)
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.

however, a function cannot be 1-many.
**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.

however, a function cannot be 1-many.

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

however, a function cannot be 1-many.

1-1 function (injection): an 1-1 relation
onto function (surjection) $f: \forall y \in Y, \exists x \in X, (x, y) \in R_f$
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.

however, 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
**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.

however, 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 \(= \{TRUE, FALSE\}\)
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 \((x, z) \in R\)
Chapter 0 Introduction

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

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

(1) reflexive: $(x, x) \in R$
**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 $(x, z) \in R$
**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
**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 graph 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 graph if \((u, v) \neq (v, u)\)
- degree: the number of edges incident on a vertex
**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 graph if \((u, v) \neq (v, u)\)
- degree: the number of edges incident on a vertex
  - in-degree and out-degree for directed graphs
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 graph if \((u, v) \neq (v, u)\)
- degree: the number of edges incident on a vertex
  in-degree and out-degree for directed graphs
- subgraph: \(H = (U, F)\) of \(G\), if \(U \subseteq V\) and \(F \subseteq E \cap (U \times U)\).
**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 graph if \((u, v) \neq (v, u)\)
- degree: the number of edges incident on a vertex
  in-degree and out-degree for directed graphs
- subgraph: \(H = (U, F)\) of \(G\), if \(U \subseteq V\) and \(F \subseteq E \cap (U \times U)\).
- path: a sequence of vertices in \(V\) connected by edges
**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 graph if \((u, v) \neq (v, u)\)
- degree: the number of edges incident on a vertex
  in-degree and out-degree for directed graphs
- subgraph: \(H = (U, F)\) of \(G\), if \(U \subseteq V\) and \(F \subseteq E \cap (U \times U)\).
- path: a sequence of vertices in \(V\) connected by edges
- simple path: the vertices on the path do not repeat
**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 graph if $(u, v) \neq (v, u)$
- Degree: The number of edges incident on a vertex
  - In-degree and out-degree for directed graphs
- Subgraph: $H = (U, F)$ of $G$, if $U \subseteq V$ and $F \subseteq E \cap (U \times U)$.
- Path: A sequence of vertices in $V$ connected by edges
- Simple path: The vertices on the path do not repeat
- Cycle: A path that starts and ends at the same vertex
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 graph if \((u, v) \neq (v, u)\)
- degree: the number of edges incident on a vertex
  in-degree and out-degree for directed graphs
- subgraph: \(H = (U, F)\) of \(G\), if \(U \subseteq V\) and \(F \subseteq E \cap (U \times U)\).
- path: a sequence of vertices in \(V\) connected by edges
- simple path: the vertices on the path do not repeat
- cycle: a path that starts and ends at the same vertex
- 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 graph if \((u, v) \neq (v, u)\)
- degree: the number of edges incident on a vertex
  - in-degree and out-degree for directed graphs
- subgraph: \(H = (U, F)\) of \(G\), if \(U \subseteq V\) and \(F \subseteq E \cap (U \times U)\).
- path: a sequence of vertices in \(V\) connected by edges
- simple path: the vertices on the path do not repeat
- cycle: a path that starts and ends at the same vertex
- tree: graph without cycles
- connected graph: there is a path between every two vertices
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** $\Sigma^* = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup \ldots$
  - is the transitive closure of $\Sigma$
- **string** is an element $\in \Sigma^*$
- **reverse of string** $w$: denoted as $w^R$
- **length of string** $w$: $|w|$, the number of symbols in $w$
- **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**: the dictionary order
- **language** $L$: a set of strings, i.e., $L \subseteq \Sigma^*$
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 \( k = 1, 2, \ldots \)

- set \( \Sigma^* = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup \ldots \) is the transitive closure of \( \Sigma \)

- string is an element \( \in \Sigma^* \)

- reverse of string \( w \): denoted as \( w^R \)

- length of string \( w \): \( |w| \), the number of symbols in \( w \)

- 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: the dictionary order

- language \( L \): a set of strings, i.e., \( L \subseteq \Sigma^* \)
Chapter 0 Introduction

String and Language

- alphabet: $\Sigma$, a finite set of symbols
- string: $s$, a finite sequence of symbols taken from an alphabet
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)
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 \( \Sigma^0 = \{ \epsilon \} \),
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 $k = 1, 2, \ldots$
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 $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cup \ldots$
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 $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$ is the transitive closure of $\Sigma$
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 $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$ is the 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 \( \Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cup \ldots \) is the transitive closure of \( \Sigma \)
• string is an element \( \in \Sigma^* \)
• reverse of string \( w \): denoted as \( w^R \)
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 $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$ is the transitive closure of $\Sigma$
- string is an element $\in \Sigma^*$
- reverse of string $w$: denoted as $w^R$
- length of string $w$: $|w|$, the number of symbols in $w$
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 $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cup \ldots$ is the transitive closure of $\Sigma$
- string is an element $\in \Sigma^*$
- reverse of string $w$: denoted as $w^R$
- length of string $w$: $|w|$, the number of symbols in $w$
- 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 \( \Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cup \ldots \) is the transitive closure of \( \Sigma \)
- string is an element \( \in \Sigma^* \)
- reverse of string \( w \): denoted as \( w^R \)
- length of string \( w \): \( |w| \), the number of symbols in \( w \)
- 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: the dictionary order
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 \( \Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots \) is the transitive closure of \( \Sigma \)
- string is an element \( \in \Sigma^* \)
- reverse of string \( w \): denoted as \( w^R \)
- length of string \( w \): \(|w|\), the number of symbols in \( w \)
- 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: the dictionary order
- language \( L \): a set of strings, i.e., \( L \subseteq \Sigma^* \)
Chapter 0 Introduction

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: \(\land\), \(\lor\), \(\neg\)
- 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
Chapter 0 Introduction

Boolean Logic:

- boolean values: $TRUE, FALSE$, or 1, 0
- boolean operands: $\land, \lor, \neg$
Chapter 0 Introduction

Boolean Logic:

- boolean values: $TRUE, FALSE$, or 1, 0
- boolean operands: $\land, \lor, \neg$
- boolean variables: $x, y, z$
Boolean Logic:

- boolean values: TRUE, FALSE, or 1, 0
- boolean operands: \( \land, \lor, \lnot \)
- 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
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
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
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 is not always easy, so:
• be patient
• be logical
• be neat/concise

Commonly seen types of proofs:
• proof by construction
• proof by contradiction
• proof by induction
Building proofs:

- be patient
- be logical
- be neat/concise

Commonly seen types of 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

commonly seen types of proofs:
Building proofs:

not always easy, so

- be patient
- be logical
- be neat/concise

commonly seen types of proofs:

- proof by construction
Building proofs:

not always easy, so

• be patient
• be logical
• be neat/concise

commonly seen types of proofs:

• proof by construction
• proof by contradiction
Building proofs:

not always easy, so

- be patient
- be logical
- be neat/concise

commonly seen types of proofs:

- proof by construction
- proof by contradiction
- proof by induction
Chapter 0 Introduction

Proof by construction

By showing explicit existence of the desired property to be proved.

For example,

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

Proof: We construct such a graph for every given $n \geq 3$.

Specifically, we construct a “ring” connecting all vertices. Formally, let $V = \{1, 2, \ldots, n\}$ be $n$ vertices. We build edges $E = \{(i, i+1) : i = 1, 2, \ldots, n-1\} \cup \{(n, 1)\}$.

Graph $G = (V, E)$ has degree 2 for every vertex in it.
Proof by construction

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

Proof: We construct such a graph for every given $n \geq 3$. Specifically, we construct a “ring” connecting all vertices. Formally, let $V = \{1, 2, ..., n\}$ be $n$ vertices. We build edges $E = \{(i, i+1) : i = 1, 2, ..., n-1\} \cup \{(n, 1)\}$ Graph $G = (V, E)$ has degree 2 for every vertex in it.
Proof by construction

By showing explicit existence of the desired property to be proved.
Proof by construction

By showing explicit existence of the desired property to be proved. For example,

Theorem 1. For every $n \geq 3$, there is a graph of $n$ vertices in which every vertex has degree 2.
Proof by construction

By showing explicit existence of the desired property to be proved. For example,

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

Proof: We construct such a graph for every given $n \geq 3$. 

Graph $G = (V,E)$ has degree 2 for every vertex in it.
Proof by construction

By showing explicit existence of the desired property to be proved. For example,

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

Proof: We construct such a graph for every given $n \geq 3$. Specifically, we construct a “ring” connecting all vertices.
Proof by construction

By showing explicit existence of the desired property to be proved. For example,

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

Proof: We construct such a graph for every given $n \geq 3$. Specifically, we construct a “ring” connecting all vertices.

Formally, let $V = \{1, 2, \ldots, n\}$ be $n$ vertices.
Proof by construction

By showing explicit existence of the desired property to be proved. For example,

Theorem 1. For every \( n \geq 3 \), there is a graph of \( n \) vertices in which every vertex has degree 2.

Proof: We construct such a graph for every given \( n \geq 3 \). Specifically, we construct a “ring” connecting all vertices.

Formally, let \( V = \{1, 2, \ldots, n\} \) be \( n \) vertices. We build edges

\[
E = \{(i, i + 1) : \text{satisfies conditions}\}.
\]
Proof by construction

By showing explicit existence of the desired property to be proved. For example,

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

Proof: We construct such a graph for every given $n \geq 3$. Specifically, we construct a “ring” connecting all vertices.

Formally, let $V = \{1, 2, \ldots, n\}$ be $n$ vertices. We build edges

$$E = \{(i, i + 1) : i = 1, 2, \ldots, n - 1\}$$
Proof by construction

By showing explicit existence of the desired property to be proved. For example,

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

Proof: We construct such a graph for every given $n \geq 3$. Specifically, we construct a “ring” connecting all vertices.

Formally, let $V = \{1, 2, \ldots, n\}$ be $n$ vertices. We build edges

$$E = \{(i, i + 1) : i = 1, 2, \ldots, n - 1\} \cup$$
Proof by construction

By showing explicit existence of the desired property to be proved. For example,

Theorem 1. For every \( n \geq 3 \), there is a graph of \( n \) vertices in which every vertex has degree 2.

Proof: We construct such a graph for every given \( n \geq 3 \). Specifically, we construct a “ring” connecting all vertices.

Formally, let \( V = \{1, 2, \ldots, n\} \) be \( n \) vertices. We build edges

\[
E = \{(i, i + 1) : i = 1, 2, \ldots, n - 1\} \cup \{(n, 1)\}
\]
Proof by construction

By showing explicit existence of the desired property to be proved. For example,

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

Proof: We construct such a graph for every given $n \geq 3$. Specifically, we construct a “ring” connecting all vertices.

Formally, let $V = \{1, 2, \ldots, n\}$ be $n$ vertices. We build edges

$$E = \{(i, i + 1) : i = 1, 2, \ldots, n - 1\} \cup \{(n, 1)\}$$

Graph $G = (V, E)$ has degree 2 for every vertex in it.
Proof by construction

Theorem 2. \( \forall n \geq 2 \text{ even}, \) there is a graph of \( n \) vertices in which every vertex has degree 3.

Proof: We construct a "ring" for all \( n \) vertices, and create one additional edge for every vertex.

Formally, let vertex set be \( V = \{1, 2, ..., n\} \).

Let edge set be \( E = \{(i, i + 1) : i = 1, 2, ..., n - 1\} \cup \{(n, 1)\} \cup \{(i, i + n/2) : i = 1, 2, ..., n/2\} \).

So, graph \( G = (V,E) \) is the desired graph.
Chapter 0 Introduction

Proof by construction

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

Proof: We construct a "ring" for all $n$ vertices, and create one additional edge for every vertex. Formally, let vertex set be $V = \{1, 2, ..., n\}$. Let edge set be $E = \{(i, i+1) : i = 1, 2, ..., n-1\} \cup \{(n, 1)\} \cup \{(i, i+n/2) : i = 1, 2, ..., n/2\}$. So, graph $G = (V, E)$ is the desired graph.
Proof by construction

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

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

Proof: We construct a "ring" for all \( n \) vertices,
Proof by construction

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

**Proof:** We construct a "ring" for all $n$ vertices, and create one additional edge for every vertex.
Proof by construction

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

**Proof:** We construct a "ring" for all $n$ vertices, and create one additional edge for every vertex.

Formally, let vertex set be $V = \{1, 2, \ldots, n\}$. 
Proof by construction

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

Proof: We construct a "ring" for all \( n \) vertices, and create one additional edge for every vertex.

Formally, let vertex set be \( V = \{1, 2, \ldots, n\} \).

Let edge set be

\[ E = \{(i, i + 1) : \]
Proof by construction

Theorem 2. \( \forall n > 2 \text{ even}, \) there is a graph of \( n \) vertices in which every vertex has degree 3.

**Proof:** We construct a ”ring” for all \( n \) vertices, and create one additional edge for every vertex.

Formally, let vertex set be \( V = \{1, 2, \ldots, n\} \).

Let edge set be

\[
E = \{(i, i + 1) : i = 1, 2, \ldots, n - 1\}
\]
Proof by construction

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

Proof: We construct a "ring" for all \( n \) vertices, and create one additional edge for every vertex.

Formally, let vertex set be \( V = \{1, 2, \ldots, n\} \).

Let edge set be

\[
E = \{(i, i + 1) : i = 1, 2, \ldots, n - 1\} \cup \{(n, 1)\}
\]
Chapter 0 Introduction

Proof by construction

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

Proof: We construct a "ring" for all \( n \) vertices, and create one additional edge for every vertex.

Formally, let vertex set be \( V = \{1, 2, \ldots, n\} \).

Let edge set be

\[
E = \{(i, i + 1) : i = 1, 2, \ldots, n - 1\} \cup \{(n, 1)\}
\]

\[
\cup \{(i, i + n/2) : i = 1, 2, \ldots, n/2\}
\]

So, graph \( G = (V, E) \) is the desired graph.
Chapter 0 Introduction

Proof by construction

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

Proof: We construct a "ring" for all \( n \) vertices, and create one additional edge for every vertex.

Formally, let vertex set be \( V = \{1, 2, \ldots, n\} \).

Let edge set be

\[
E = \{(i, i + 1) : i = 1, 2, \ldots, n - 1\} \cup \{(n, 1)\}
\]

\[ \cup \{(i, i + n/2) : i = 1, 2, \ldots, n/2\} \]

So, graph \( G = (V, E) \) is the desired graph.
Chapter 0 Introduction

Proof by contradiction

By assuming the opposite to draw a contradiction.

Theorem 3. (Pigeonhole principle). Placing \( n \) pigeons in \( k \) holes, \( k < n \), then there is at least one hole hosting more than one pigeon.

Proof: Assume otherwise, i.e., assume at most one pigeon in every hole. Then the total number of pigeons is \( \leq k < n \). This contradicts to the fact that there are \( n \) pigeons. So the assumption of at most one pigeon in every hole was incorrect.
Chapter 0 Introduction

**Proof by contradiction**
Proof by contradiction

By assuming the opposite to draw a contradiction.
Proof by contradiction

By assuming the opposite to draw a contradiction.

Theorem 3. (Pigeonhole principle). Placing $n$ pigeons in $k$ holes, $k < n$, then there is at least one hole hosting more than one pigeon.
Proof by contradiction

By assuming the opposite to draw a contradiction.

Theorem 3. (*Pigeonhole principle*). Placing $n$ pigeons in $k$ holes, $k < n$. then there is at least one hole hosting more than one pigeon.

**Proof:** Assume otherwise,
Proof by contradiction

By assuming the opposite to draw a contradiction.

Theorem 3. (*Pigeonhole principle*). Placing \( n \) pigeons in \( k \) holes, \( k < n \), then there is at least one hole hosting more than one pigeon.

*Proof*: Assume otherwise, i.e., assume at most one pigeon in every hole.
Proof by contradiction

By assuming the opposite to draw a contradiction.

Theorem 3. *(Pigeonhole principle)*. Placing $n$ pigeons in $k$ holes, $k < n$. then there is at least one hole hosting more than one pigeon.

**Proof**: Assume otherwise, i.e., assume at most one pigeon in every hole.

Then the total number of pigeons is $\leq k$: 
Proof by contradiction

By assuming the opposite to draw a contradiction.

Theorem 3. (*Pigeonhole principle*). Placing \( n \) pigeons in \( k \) holes, \( k < n \). then there is at least one hole hosting more than one pigeon.

Proof: Assume otherwise, i.e., assume at most one pigeon in every hole.

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

Proof by contradiction

By assuming the opposite to draw a contradiction.

Theorem 3. (*Pigeonhole principle*). Placing \( n \) pigeons in \( k \) holes, \( k < n \), then there is at least one hole hosting more than one pigeon.

**Proof:** Assume otherwise, i.e., assume at most one pigeon in every hole.

Then the total number of pigeons is  \( \leq k < n \).

This contradicts to the fact that there are \( n \) pigeons.
Proof by contradiction

By assuming the opposite to draw a contradiction.

Theorem 3. (*Pigeonhole principle*). Placing \(n\) pigeons in \(k\) holes, \(k < n\). then there is at least one hole hosting more than one pigeon.

**Proof**: Assume otherwise, i.e., assume at most one pigeon in every hole.

Then the total number of pigeons is \(\leq k < n\).

This contradicts to the fact that there are \(n\) pigeons.

So the assumption of at most one pigeon in every hole was incorrect.
Chapter 0 Introduction

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

**(Proof)**: Assume otherwise, i.e., \(\sqrt{2} = \frac{m}{n}\) where at least one of \(m\) and \(n\) is odd.

Using the assumption, we have \(n\sqrt{2} = m\); square both sides, we have \(2n^2 = m^2\); that \(m^2\) is even implies that \(m\) is even. (**exercise**) \(m\) can be written as \(m = 2k\), for some integer \(k\).

And we know \(2n^2 = m^2 = 4k^2\), that is, \(n^2 = 2k^2\). Thus, \(n^2\) is even; this implies \(n\) is even.

Both \(n\) and \(m\) now are even. This contradicts the assumption that at least one of \(m\) and \(n\) is odd.

The assumption that \(\sqrt{2}\) is rational was incorrect!
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 $n$.)
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,
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., $\sqrt{2} = m/n$
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., $\sqrt{2} = m/n$
where at least one of $m$ and $n$ is odd.

Using the assumption, we have $n \sqrt{2} = m$; square both sides, we have $2n^2 = m^2$;
that $m^2$ is even implies that $m$ is even. (exercise)
$m$ can be written as $m = 2k$, for some integer $k$.
And we know $2n^2 = m^2 = 4k^2$,
that is, $n^2 = 2k^2$; this implies $n$ is even.
Both $n$ and $m$ now are even.
This contradicts the assumption that at least one of $m$ and $n$ is odd.
The assumption that $\sqrt{2}$ is rational was incorrect!
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., \( \sqrt{2} = m/n \)
where at least one of \( m \) and \( n \) is odd.

Using the assumption, we have \( n\sqrt{2} = m \);
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., $\sqrt{2} = m/n$
where at least one of $m$ and $n$ is odd.

Using the assumption, we have $n\sqrt{2} = m$;
square both sides, we have $2n^2 = m^2$;
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., $\sqrt{2} = m/n$
where at least one of $m$ and $n$ is odd.

Using the assumption, we have $n\sqrt{2} = m$;
square both sides, we have $2n^2 = m^2$;
that $m^2$ is even
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., \( \sqrt{2} = m/n \) where at least one of \( m \) and \( n \) is odd.

Using the assumption, we have \( n\sqrt{2} = m \);

square both sides, we have \( 2n^2 = m^2 \);
that \( m^2 \) is even implies that \( m \) is even.
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., $\sqrt{2} = m/n$
where at least one of $m$ and $n$ is odd.

Using the assumption, we have $n\sqrt{2} = m$;
square both sides, we have $2n^2 = m^2$;
that $m^2$ is even implies that $m$ is even. (exercise)
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., $\sqrt{2} = m/n$ where at least one of $m$ and $n$ is odd.

Using the assumption, we have $n\sqrt{2} = m$;

square both sides, we have $2n^2 = m^2$;
that $m^2$ is even implies that $m$ is even. (exercise)

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

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

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., $\sqrt{2} = m/n$ where at least one of $m$ and $n$ is odd.

Using the assumption, we have $n\sqrt{2} = m$;
square both sides, we have $2n^2 = m^2$;
that $m^2$ is even implies that $m$ is even. (exercise)

$m$ can be written as $m = 2k$, for some integer $k$.
And we know $2n^2 = m^2$
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., $\sqrt{2} = m/n$ where at least one of $m$ and $n$ is odd.

Using the assumption, we have $n\sqrt{2} = m$;

square both sides, we have $2n^2 = m^2$;
that $m^2$ is even implies that $m$ is even. (exercise)

$m$ can be written as $m = 2k$, for some integer $k$.
And we know $2n^2 = m^2 = 4k^2$,
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., $\sqrt{2} = m/n$ where at least one of $m$ and $n$ is odd.

Using the assumption, we have $n\sqrt{2} = m$;

square both sides, we have $2n^2 = m^2$;
that $m^2$ is even implies that $m$ is even. (exercise)

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

And we know $2n^2 = m^2 = 4k^2$,
that is, $n^2 = 2k^2$
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., $\sqrt{2} = m/n$ where at least one of $m$ and $n$ is odd.

Using the assumption, we have $n\sqrt{2} = m$;

square both sides, we have $2n^2 = m^2$;

that $m^2$ is even implies that $m$ is even. (exercise)

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

And we know $2n^2 = m^2 = 4k^2$,

that is, $n^2 = 2k^2$

Thus, $n^2$ is even; this implies $n$ is even.
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., \( \sqrt{2} = m/n \) where at least one of \( m \) and \( n \) is odd.

Using the assumption, we have \( n\sqrt{2} = m \);
square both sides, we have \( 2n^2 = m^2 \);
that \( m^2 \) is even implies that \( m \) is even. (exercise)

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

And we know \( 2n^2 = m^2 = 4k^2 \),
that is, \( n^2 = 2k^2 \)

Thus, \( n^2 \) is even; this implies \( n \) is even.
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., $\sqrt{2} = m/n$
where at least one of $m$ and $n$ is odd.

Using the assumption, we have $n\sqrt{2} = m$;
square both sides, we have $2n^2 = m^2$;
that $m^2$ is even implies that $m$ is even. (exercise)

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

And we know $2n^2 = m^2 = 4k^2$,
that is, $n^2 = 2k^2$

Thus, $n^2$ is even; this implies $n$ is even.

Both $n$ and $m$ now are even.
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., $\sqrt{2} = m/n$ where at least one of $m$ and $n$ is odd.

Using the assumption, we have $n\sqrt{2} = m$;

square both sides, we have $2n^2 = m^2$;
that $m^2$ is even implies that $m$ is even. (exercise)

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

And we know $2n^2 = m^2 = 4k^2$,
that is, $n^2 = 2k^2$

Thus, $n^2$ is even; this implies $n$ is even.

Both $n$ and $m$ now are even.
This contradicts the assumption that at least one of $m$ and $n$ is 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., $\sqrt{2} = m/n$
where at least one of $m$ and $n$ is odd.

Using the assumption, we have $n\sqrt{2} = m$;
square both sides, we have $2n^2 = m^2$;
that $m^2$ is even implies that $m$ is even. (exercise)

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

And we know $2n^2 = m^2 = 4k^2$,
that is, $n^2 = 2k^2$

Thus, $n^2$ is even; this implies $n$ is even.

Both $n$ and $m$ now are even.
This contradicts the assumption that at least one of $m$ and $n$ is odd.
The assumption that $\sqrt{2}$ is rational was incorrect!
Chapter 0 Introduction

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
proof by induction

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

- The first domino will be pushed down, and
- 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 $n = 1$,
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.
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.,

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

\[\]
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 $P(n)$ 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
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

We need to prove:

\[ P(1) \Rightarrow P(k+1) \]

for general \( k \).
Chapter 0 Introduction

We need to prove:

\[ P(1) \Rightarrow P(2) \]

\[ P(k) = \text{ "the } k\text{th domino will fall"} \]

\[ P(2) \Rightarrow P(3) \]

\[ P(6) \Rightarrow P(7) \]

\[ P(1) \Rightarrow P(2) \]
Chapter 0 Introduction

We need to prove:

\[ P(k) \equiv \text{“the } k\text{th domino will fall”} \]

\[ P(1) \Rightarrow P(2) \]
\[ P(2) \Rightarrow P(3) \]
\[ P(3) \Rightarrow P(4) \]
\[ P(4) \Rightarrow P(5) \]
\[ P(5) \Rightarrow P(6) \]
\[ P(6) \Rightarrow P(7) \]

\[ P(7) \Rightarrow P(8) \]
We need to prove:

(1) $P(1)$
Chapter 0 Introduction

We need to prove:

(1) $P(1)$

(2) $P(k) \implies P(k + 1)$ for general $k$. 

$P(k) \equiv \text{“the kth domino will fall”}$
Theorem 5. 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. When $n = 1$, $P(n)$ holds because $1 = \frac{1}{2}(1+1)$.

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

We now show $P(n)$ holds for $n = k + 1$ in the following:

$1 + 2 + \cdots + k + k + 1 = (1 + 2 + \cdots + k) + (k + 1) = \frac{k}{2}(k+1) + (k + 1) = (k+1)\left(\frac{k}{2} + 1\right) = (k+1)\left(\frac{k+2}{2}\right) = (k+1)^2/2$.

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

Chapter 0 Introduction

Theorem 5. For all $n \geq 1$, summation $1 + 2 + \cdots + n = \frac{n}{2}(n + 1)$

Chapter 0 Introduction

Theorem 5. For all $n \geq 1$, summation $1 + 2 + \cdots + n = \frac{n}{2}(n + 1)$.
We use proof by induction to prove.
Chapter 0 Introduction

Theorem 5. 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?
Chapter 0 Introduction

Theorem 5. 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?

$P(n) \equiv \text{“}1 + 2 + \cdots + n = \frac{n}{2}(n + 1)\text{”}$
Chapter 0 Introduction

Theorem 5. 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?

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

Proof.
Chapter 0 Introduction

Theorem 5. 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. when \( n = 1 \),
Chapter 0 Introduction

Theorem 5. 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. when \( n = 1 \), \( P(n) \) holds because \( 1 = \frac{1}{2}(1 + 1) \).
Theorem 5. 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 "1 + 2 + \cdots + n = \frac{n}{2}(n + 1)"

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

Assume $\mathcal{P}(n)$ holds when $n = k$, 
Chapter 0 Introduction

Theorem 5. 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?

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

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

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

Theorem 5. 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. When \( n = 1 \), \( P(n) \) holds because \( 1 = \frac{1}{2}(1 + 1) \).

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

We now show \( P(n) \) holds for \( n = k + 1 \) in the following:
Chapter 0 Introduction

Theorem 5. 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. when $n = 1$, $P(n)$ holds because $1 = \frac{1}{2}(1 + 1)$.

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

We now show $P(n)$ holds for $n = k + 1$ in the following:

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

Theorem 5. 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?

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

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

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

We now show \( \mathcal{P}(n) \) holds for \( n = k + 1 \) in the following:

\[
1 + 2 + \cdots + k + k + 1 = (1 + 2 + \cdots + k) + (k + 1)
= \frac{k}{2}(k + 1) + (k + 1) \text{ by the assumption}
\]
Theorem 5. 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. when \( n = 1 \), \( P(n) \) holds because \( 1 = \frac{1}{2}(1 + 1) \).

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

We now show \( P(n) \) holds for \( n = k + 1 \) in the following:

\[
1 + 2 + \cdots + k + k + 1 = (1 + 2 + \cdots + k) + (k + 1) \\
= \frac{k}{2}(k + 1) + (k + 1) \text{ by the assumption} \\
= (k + 1)(\frac{k}{2} + 1)
\]
Chapter 0 Introduction

Theorem 5. 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 "1 + 2 + \cdots + n = \frac{n}{2}(n + 1)"

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

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

We now show $\mathcal{P}(n)$ holds for $n = k + 1$ in the following:

$1 + 2 + \cdots + k + k + 1 = (1 + 2 + \cdots + k) + (k + 1)$
$= \frac{k}{2}(k + 1) + (k + 1) \text{ by the assumption}$
$= (k + 1)(\frac{k}{2} + 1)$
$= (k + 1)(\frac{k}{2} + \frac{2}{2})$
Theorem 5. 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?

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

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

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

We now show \( P(n) \) holds for \( n = k + 1 \) in the following:

\[
1 + 2 + \cdots + k + k + 1 = (1 + 2 + \cdots + k) + (k + 1) \\
= \frac{k}{2}(k + 1) + (k + 1) \text{ by the assumption} \\
= (k + 1)\left(\frac{k}{2} + 1\right) \\
= (k + 1)\left(\frac{k}{2} + \frac{2}{2}\right) \\
= \frac{k+1}{2}(k + 1) + 1
\]

We have proved \( P(n) \) holds for all \( n \geq 1 \).
Chapter 0 Introduction

Theorem 5. 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?

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

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

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

We now show $P(n)$ holds for $n = k + 1$ in the following:

\[
1 + 2 + \cdots + k + k + 1 = (1 + 2 + \cdots + k) + (k + 1) = \frac{k}{2}(k + 1) + (k + 1) \text{ by the assumption} \\
= (k + 1)(\frac{k}{2} + 1) \\
= (k + 1)(\frac{k}{2} + \frac{2}{2}) \\
= \frac{k+1}{2}((k + 1) + 1)
\]

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