The United Nations University

UNU/IIST

International Institute for Software Technology

Mathematics for Computer Science

Yumbayar Namsrai

August 1999

UNU/IIST Report No. 173

UNU/IIST and UNU/IIST Reports

UNU/IIST is a Research and Training Center of the United Nations University. It was founded in 1992, and is located in Macau. UNU/IIST is jointly funded by the Governor of Macau and the Governments of China and Portugal through contribution to the UNU Endowment Fund.

The mission of UNU/IIST is to assist developing countries in the application and development of software technology.

UNU/IIST contributes through its programmatic activities:

1. advanced development projects in which software techniques supported by tools are applied,
2. research projects in which new techniques for software development are investigated,
3. curriculum development projects in which courses of software technology for universities in developing countries are developed,
4. courses which typically teach advanced software development techniques,
5. events in which conferences and workshops are organised or supported by UNU/IIST, and
6. dissemination, in which UNU/IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU/IIST is on formal methods for software development. UNU/IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU/IIST produces a report series. Reports are either Research R, Technical T, Compendia C or Administrative A. They are records of UNU/IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU/IIST or visit UNU/IIST home page: http://www.iist.unu.edu, if you would like to know more about UNU/IIST and its report series.

Zhou Chaochen, Director — 01.8.1997 – 31.7.2001

The United Nations University

UNU/IIST

International Institute for Software Technology

P.O. Box 3058
Macau

Mathematics for Computer Science

Yumbayar Namsrai

Abstract

To give students a solid and rigorous background in computer science the requisite mathematical foundations are necessary. Mathematical logic (propositional and predicate logics), set theory, mappings, relations, and recursive and inductive definition of processes and properties of a system are very important especially for courses such as "Software Specification", "Formal Methods for Software Development" and "Reasoning About Software". The use of programming languages which admit recursive routines requires precise understanding of recursion. We present detailed teaching materials for these subjects of mathematics.

Yumbayar Namsrai was a UN Fellow at UNU/IIST from September 1996 to June 1997 and from May 1999 to August 1999. He studied mathematics at The National University of Mongolia in Ulaanbaatar, Mongolia, from 1967 to 1972, and worked at the Joint Institute for Nuclear Research (JINR), Dubna, USSR from 1972 to 1981 where he was awarded a Candidate of Science degree in Physics and Mathematics (Software Systems of Computers and Computing Systems) in 1982. After returning to Mongolia he became the Head of the Programming Department in the Mathematics Faculty at The National University of Mongolia, and in January 1998 moved to the Computer Science and Management School of the Mongolian Technical University, also in Ulaanbaatar, where he is now Head of the Computer Science Department. His research interests are in the computerisation of the traditional Mongolian script.

Contents

1 Introduction

According to [1], mathematical maturity is essential to the successful mastery of several fundamental topics in computing, and all computing students should take mathematics courses which cover at least the following subjects:

Discrete mathematics: sets, functions, elementary propositional and predicate logic, elementary graph theory, proof theory, combinatorics, probability, and random numbers.

Calculus: differential and integral calculus, including sequences and series and an introduction to differential equations.

In addition, mathematics courses should include some of the following subjects:

Probability: discrete and continuous, including elementary statistics.

Linear algebra: elementary, including matrices, vectors, and linear transformations.

Mathematical logic: propositional and functional calculi, completeness, validity, proof, and decision problems.

Based on these guidelines and on an extensive investigation of the curricula of computer science departments of several universities (where mathematics courses for computing are variously called "Discrete mathematics", "Discrete structures", "Discrete mathematics in computer science", "Discrete structures in computer science", "Mathematics of computer science", etc.) we propose a course "Mathematics for computer science" with the following contents:

1. Numbers
   Positional and based number systems; decimal, binary, octal and hexadecimal systems; radix conversion; representation of numbers in computers: natural numbers, two's complement, signed integers and floating-point numbers; least common multiple and greatest common divisor of positive integers, and algorithms for computing them; primes and factorisation; congruence and modular arithmetic.
2. Classical Logic
   - 2.1 Propositional logic: propositions and logical connectives; propositional expressions; rules of precedence; evaluation of expressions; truth tables; properties of expressions; logical equivalences; proof techniques.
   - 2.2 Predicate logic: introduction to predicate logic; truth sets of predicates; connections of predicates; quantifiers; expressions of predicate logic; validity of expressions.

3. Arguments and proof techniques
   Argument forms; validity of an argument and testing arguments for validity; valid argument forms: modus ponens and modus tollens, disjunctive addition, conjunctive simplification, disjunctive syllogism and hypothetical syllogism; proof techniques: vacuous proof, trivial proof, direct proof, proof of the contrapositive, proof by contradiction, proof by cases.

4. Set theory
   Introduction to set theory; relationships between sets; operations on sets; laws of set algebra.

5. Mappings
   Introduction to mappings; relationships between mappings; operations on mappings.

6. Functions
   Introduction to functions; defining functions; classification of functions; composition of functions.

7. Relations
   Introduction to relations; functions and relations; classification of binary relations; operations on relations.

8. Recursion
   Introduction to recursion; examples.

9. Induction
   Introduction to induction; proof by induction; first and second principles of induction.

10. Counting, Permutations and Combinations
    Basics of counting; permutations; combinations; formulae involving combinations; inclusion and exclusion principle; the pigeon hole principle.

11. Graph theory
    Introduction to graph theory; connected graphs; isomorphisms and subgraphs; matrix representations of graphs; weighted graphs; Warshall's algorithm for computing the existence of paths; Dijkstra's algorithm for finding the shortest path; trees; rooted trees; binary trees.

12. Random numbers
    Introduction to random numbers; the expected value; the chi-square test.

In this report we present material which could form the contents of those seven of the above topics which are the most important mathematical background for students of formal methods. In particular, this material could form the basis for an introductory course on mathematics for those who wish to study formal methods but who lack the required mathematical knowledge to attend a formal methods course directly, and as such could be given prior to the formal methods course, for example the course on RAISE (Rigorous Approach to Industrial Software Engineering) [2, 3] given by UNU/IIST.

The report also includes a section which gives an introduction to propositional modal logic. This topic does not appear in the list above, but it forms the mathematical basis for much research work in formal methods, in particular for the Duration Calculus which is one of UNU/IIST's main research areas. This section could be used as introductory material by those wishing to study Duration Calculus, and could also form the basis for a short introductory course given prior to a course on Duration Calculus for those students who are not familiar with modal logic.

Although we focus on the topics which are most important for formal methods, the material presented, suitably extended with material for the sections not treated here, could of course form the bulk of a more general "Mathematics for computer science" course for university computer science departments, and as such could be useful to university computer science students in general.

Teaching materials for the sections included in this report, specifically overhead projector foils for lecturers, are available from UNU/IIST and can in fact be downloaded electronically from UNU/IIST's home pages at the following URL:
http://www.iist.unu.edu/home/Unuiist/newrh/II/1/3/2/page.html.

2 Classical logic

Logic is the science of order and form. Even if we do not know whether there is a zoo in Macau

$$\text{There is a zoo in Macau } \textit{or} \text{ There is not a zoo in Macau}$$

is true. The truth of the sentence can be determined from its structure all without knowing whether its constituents are true or false. Similarly, we determine that the sentence

$$\pi > 1.0 \textit{ or } \pi \leq 1.0$$

is true without knowing the value of π. In fact, both sentences are instances of the abstract sentence

$$P \textit{ or } (\text{not } P)$$

and any sentence of this form is true regardless of whether P is a true or a false proposition. Consider the two following statements:

If I wake up late or if I miss the bus, I will be late for work.
Therefore, if I arrived at work on time, I did not wake up late and I did not miss the bus.

If x is a real number such that $x < -2$ or $2 < x$, then $x^2 > 4$.
Therefore, if $x^2 < 4$, then $-2 < x$ and $x < 2$.

Logic helps us to analyse an argument's form to determine if the truth of the conclusion follows from the truth of the preceding statements. While the content of the two statements above is different, their logical form is similar.

Let P stand for the statements 'I wake up late' and 'x is a real number such that $x < -2$'. Let Q stand for the statements 'I miss the bus' and 'x is a real number such that $x > 2$'. Let R stand for the statements 'I am late for work' and '$x^2 > 4$'. Then the common form of both of the above arguments is:

If P or Q then R.
Therefore, if (not R) then (not P) and (not Q).

Logic also provides a notation for specifications that is precise, comprehensive, economical and easy to manipulate. This chapter presents a language of abstract sentences, called propositional logic, and introduces techniques for determining whether a given abstract sentence is valid or contradictory and whether two given abstract sentences are equivalent.

2.1 Propositional Logic

2.1.1 Propositions and Logical Connectives

The building blocks for logical constructs (or logical sentences) are the set of all declarative sentences which can be classified as true or false, but not both. We call such a declarative sentence a *proposition* (or a *statement*). For example, 'I am in Macau' is a statement, but 'I am lying' is not a statement because it is not possible to say whether it is true or false. Similarly, 'A (male) barber who shaves only men who do not shave themselves, shaves himself' is not a statement. According to the second clause of this sentence, the barber shaves himself. But the first clause states that he only shaves men who do not shave themselves, therefore he does not shave himself. This is contradictory.

This type of sentence is known as a paradox.

Propositions may be elementary, in the sense that their truth is trivially known, or *compound sentences*. These compound sentences are built from elementary ones which are joined or mod-

ified using a set of well-defined connectives or operators.

One goal of logic is to define a set of connectives and how they operate, and to determine the truth value of compound propositions (sentences).

We first introduce the basic symbols and show how they are combined to form the sentences of propositional logic.

Definition (propositions)
The sentences of propositional logic are made up of the following symbols, called propositions:

- The truth constants *true* and *false*
  We sometimes use T instead of *true* and F instead of *false* for simplicity.
- The propositional letters
  $P, Q, R, S, P_1, Q_1, R_1, S_1, P_2, ...$ (the capital letters, possibly with a numerical subscript)

Definition (connectives) Let P and Q be statements.

1. **AND**
   - The proposition 'P and Q', denoted by $P \wedge Q$, is true if both P and Q are true and false otherwise.
   - It is called the conjunction of P and Q.
   - Examples
     (a) '2+2=4' $\wedge$ '3−2=1' is true.
     (b) 'It is raining' $\wedge$ 'It is not raining' is false.
2. **OR**
   - The proposition 'P or Q', denoted by $P \vee Q$, is false only if both P and Q are false; otherwise it is true.
   - It is called the disjunction of P and Q.
   - Examples
     (a) '2+2=4' $\vee$ 'I am sitting' is true.
     (b) 'It is raining' $\vee$ 'It is not raining' is true.
     (c) '$9 - 1 < 7$' $\vee$ '$10 - 1 < 8$' is false.
3. **NOT**
   - The proposition 'not P', denoted by $\sim P$, is true if P is false and false if P is true.

- $\sim P$ is called the negation of P.
- P is true if $\sim P$ is false and conversely.
- Examples
  (a) not '2+2=4' (or $\sim$'2+2=4') is false.
  (b) $\sim (9 - 1 < 7 \ \vee \ 10 - 1 < 8)$ is true.

4. **CONDITIONAL**

- A conditional statement is a statement of the form 'if P, then Q'.
- It is written as $P \rightarrow Q$, P being called the antecedent and Q the consequent.
- $P \rightarrow Q$ is false only if P is true and Q is false, otherwise it is true.
- Examples
  (a) The sentence 'If today is Friday, then 3+2=6' would be written in the form $P \rightarrow Q$, where P stands for the sentence 'Today is Friday' and Q stands for the sentence '3+2=6'. It is true if today is not Friday, but if today is Friday it is false because Q is false.
  (b) 'If it is sunny today, then we will go to the beach' is true if it is sunny and we go to the beach, but it is false if it is sunny and we do not go to the beach.

5. **EQUIVALENCE** or **BICONDITIONAL**

- A statement of the form: 'P if and only if Q' is called the equivalence or the biconditional of P and Q. It is written $P \equiv Q$ or $P \leftrightarrow Q$ or P iff Q.
- $P \equiv Q$ is true only if P and Q have the same truth value.
- Examples
  (a) '1+1=2' $\equiv$ '5−3=2' is true.
  (b) 'The earth is flat' $\equiv$ '3 < 5' is false.

The truth values of the connectives are summarised in the following *truth tables*:

P	$\sim P$
T	F
F	T

Here, if the truth value of P is as given in the first column, the truth value of $(\sim P)$ is as shown in the final column.

P	Q	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \equiv Q$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

Here, if the truth values of P and Q are as given in the first two columns, the truth values of the sentences $(P \wedge Q)$, $(P \vee Q)$, ... are as shown in the appropriate columns.

Observe that according to the definition of the **OR** connective, it is "inclusive" in the sense that $P \vee Q$ is true when both P and Q are true.

Note also that, according to the definition of the **CONDITIONAL** connective, the sentence $(P \rightarrow Q)$ is true whenever its antecedent P is false or its consequent Q is true i.e. $\mathit{false} \rightarrow P$ and $P \rightarrow \mathit{true}$ are true for any sentence P.

In logic, the sentence $P \rightarrow Q$ is well-defined even if there is no causal relationship between P and Q. For example, the concrete sentence

If Macau is the capital of England, then it is summer here

is considered to be a true sentence irrespective of whether or not it is summer because its antecedent is false.

2.1.2 Propositional Expressions

Expressions of propositional logic are built up from propositions by application of the propositional connectives. We use the script letters $\mathcal{E}$, $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$, possibly with a numerical subscript, to stand for expressions.

Definition (expressions)
Expressions are formed according to the following rules:

- Every proposition, i.e. a truth constant or a propositional letter, is an expression.
- If $\mathcal{F}$ and $\mathcal{G}$ are expressions, then so are the following connections of them:

$$(\sim \mathcal{F})$$
$$(\sim \mathcal{G})$$
$$(\mathcal{F} \wedge \mathcal{G})$$
$$(\mathcal{F} \vee \mathcal{G})$$
$$(\mathcal{F} \rightarrow \mathcal{G})$$
$$(\mathcal{F} \equiv \mathcal{G})$$

Example
The following strings are expressions:

$$((\sim (P \vee Q)) \equiv ((\sim P) \wedge (\sim Q)))$$
$$(((P \vee Q) \wedge (Q \rightarrow R)) \rightarrow ((P \wedge R) \rightarrow (\sim R)))$$

For example,

$$\mathcal{F}: \ ((\sim (P \vee Q)) \equiv ((\sim P) \wedge (\sim Q)))$$

is an expression because both P and Q are expressions, hence

$$(P \vee Q), (\sim P), \text{ and } (\sim Q)$$

are expressions, hence

$$(\sim (P \vee Q)) \text{ and } ((\sim P) \wedge (\sim Q))$$

are expressions, hence the given

$$\mathcal{F} : ((\sim (P \vee Q)) \equiv ((\sim P) \wedge (\sim Q)))$$

is an expression.

2.1.3 Rules of Precedence

According to the definition of propositional expressions, any propositional expression should be of the form $\sim (Expr)$ or $(Expr_1 * Expr_2)$, where '*' is one of the binary logical connectives. This makes propositional expressions very large, in the sense that they are written with many pairs of parentheses. To keep the number of parentheses to a minimum, some rules of precedence are introduced which guarantee a unique meaning for a compound proposition.

Definition (precedence rules)

1. The order of evaluation is $\sim \wedge \vee \rightarrow \equiv$ ($\sim$ has the highest priority).
2. We may always omit the outermost pair of parentheses.
3. If there are different connectives in an expression, then that connective which has the highest priority evaluates first, and if more than one occurrence of the same connective is in an expression, then these connectives evaluate from left to right.

4. If there are more than one successive occurrence of the same connective in an expression, then we may only omit the outermost pair of parentheses of the subexpression on the left side of each connective.

5. We may omit all pairs of parentheses which can unambiguously be restored using the above four rules.

Examples

$$\mathcal{F} : ((\sim (P \vee Q)) \equiv ((\sim P) \wedge (\sim Q)))$$

can be written as follows:

Expressions	Rules used
$\mathcal{F} : (\sim (P \vee Q)) \equiv ((\sim P) \wedge (\sim Q))$	rule 2
$\mathcal{F} : \sim (P \vee Q) \equiv (\sim P) \wedge (\sim Q)$	the outermost parentheses of subexpressions on both sides of $\equiv$ are omitted
$\mathcal{F} : \sim (P \vee Q) \equiv \sim P \wedge \sim Q$	the parentheses of subexpressions on both sides of $\wedge$ are omitted

The expression

$$\mathcal{E} : ((P \to Q) \to (R \to S))$$

can be written as:

$$\mathcal{E} : (P \to Q) \to (R \to S) \qquad \textit{(rule 2)}$$

There are three occurrences of $\to$ in the expression. There is no subexpression in parentheses on the left of the second or the third of them, but subexpressions on both sides of the second are in parentheses. According to rule 4, we may omit the parentheses of the subexpression on the left:

$$\mathcal{E} : P \to Q \to (R \to S) \qquad \textit{(rule 4)}$$

NOTATION

Of course, there may be parentheses in an expression that cannot be omitted without losing the meaning of the original expression. For example, the parentheses in the following expressions cannot be omitted.

$$\mathcal{E} : P \to Q \to (R \to S)$$
$$\mathcal{F} : \sim (P \vee Q) \equiv \sim P \wedge \sim Q$$

Let us insert parentheses in the expression

$$\mathcal{E} : P \to Q \to R \to S$$

There are three $\to$ connectives, and a propositional letter (P) is on the left of the first (from the left) connective, so it should not be in parentheses. There are subexpressions on both sides of the second one, but according to rule 4 only parentheses of the left subexpression may be omitted, so the expression is equivalent to the following expression:

$$\mathcal{E} : (P \to Q) \to R \to S$$

Now there is a subexpression on the left of the third conditional sign and its parentheses may be omitted, so we arrive at the following form:

$$\mathcal{E} : ((P \to Q) \to R) \to S$$

Finally, we write the whole expression in parentheses:

$$\mathcal{E} : (((P \to Q) \to R) \to S)$$

We can check that this is not equivalent to $((P \to Q) \to (R \to S))$ which is the original expression we started with.

Let us similarly insert parentheses in the expression

$$\mathcal{F} : \sim P \vee Q \equiv \sim P \wedge \sim Q$$

First, all negations can be in parentheses because negation has the highest priority:

$$\mathcal{F} : (\sim P) \vee Q \equiv (\sim P) \wedge (\sim Q)$$

Next, the conjunction may be in parentheses because it has the second highest priority:

$$\mathcal{F} : (\sim P) \vee Q \equiv ((\sim P) \wedge (\sim Q))$$

Then, the disjunction may be in parentheses because it has higher priority than equivalence:

$$\mathcal{F} : ((\sim P) \vee Q) \equiv ((\sim P) \wedge (\sim Q))$$

Finally, we arrive at

$$\mathcal{F} : (((\sim P) \vee Q) \equiv ((\sim P) \wedge (\sim Q)))$$

which is not the same as $((\sim (P \vee Q)) \equiv ((\sim P) \wedge (\sim Q)))$.

2.1.4 Evaluating Expressions

For any expression $\mathcal{E}$, we often need to calculate its truth value by assigning a given truth value, either *true* or *false*, to each of the propositional letters in the expression. To do this, let n be the number of propositional letters contained in the expression. Then we may use the following algorithm.

Evaluation algorithm

1. Create a table with two rows and $n+1$ columns and write the propositional letters of the given expression in the first n columns of the first row and the expression itself in the last column.
2. Number all the operators (connectives) within the expression according to their evaluation order as defined by the precedence rules. Here we should recall that the subexpression in the innermost parentheses must be evaluated first.
3. Fill the columns of the second row with the given truth values of the corresponding propositional letters.
4. Evaluate the operator which has the lowest number which has not yet been evaluated, and write its result in the second row, exactly under the sign of the operator.
5. Repeat the previous step until the last operator has been evaluated.

Examples
Evaluate the expression

$$\mathcal{E} : (P \to Q) \wedge (Q \to P)$$

when P is false and Q is true. After the first three steps of the algorithm, the table would be:

		1	*3*	*2*
P	Q	$(P \to Q)$	$\wedge$	$(Q \to P)$
F	T			

In the next step, we should evaluate the operator which has number 1. Its result is true according to the truth table of $\to$:

		1	*3*	*2*
P	Q	$(P \to Q)$	$\wedge$	$(Q \to P)$
F	T	T		

Now, we should evaluate operator number 2. Its result is false:

		1	*3*	*2*
P	Q	$(P \to Q)$	$\wedge$	$(Q \to P)$
F	T	T		F

Finally, we evaluate the last operator. Its result is false and this is therefore the value of the whole expression for the given values of its propositional letters P and Q:

		1	*3*	*2*
P	Q	$(P \to Q)$	$\wedge$	$(Q \to P)$
F	T	T	F	F

2.1.5 Truth Tables

For a given expression, a table containing its values under all possible combinations of truth values of the propositional letters within the expression is called the *truth table* of the given expression.

If an expression contains only two propositional letters P and Q, we distinguish two cases, assigning P the truth values true and false respectively. In each case we also distinguish between two sub-cases, assigning Q the truth values true and false respectively. Thus, for an expression containing only two propositional letters P and Q, there are four possible combinations of truth values of the letters. So we must consider the following four pairs:

P	Q
T	T
T	F
F	T
F	F

In general, if an expression contains n propositional letters there are 2^n different combinations of truth values of the letters.

We can use the algorithm described above to calculate the truth table of a given expression. Thus, if an expression contains n propositional letters, then its truth table contains $n+1$ rows and $n+1$ columns.

Examples
The truth table of the expression

$$\mathcal{E} : (P \to Q) \wedge (Q \to P)$$

is

		1	*3*	*2*
P	Q	$(P \to Q)$	$\wedge$	$(Q \to P)$
T	T	T	T	T
T	F	F	F	T
F	T	T	F	F
F	F	T	T	T

Now, suppose our given expression is

$$\mathcal{F} : \sim (P \vee Q) \equiv \sim P \wedge \sim Q.$$

Its truth table is

		2	*1*	*6*	*3*	*5*	*4*
P	Q	$\sim$	$(P \vee Q)$	$\equiv$	$\sim P$	$\wedge$	$\sim Q$
T	T	F	T	T	F	F	F
T	F	F	T	T	F	F	T
F	T	F	T	T	T	F	F
F	F	T	F	T	T	T	T

2.1.6 Properties of expressions

1. **Definition (tautology)**
   An expression $\mathcal{E}$ is called a *tautology* if it is always true regardless of the truth values of its propositional letters (or component letters).

2. **Definition (contradiction)**
   A *contradiction* is an expression which is false regardless of the truth values of its propositional letters. So, if $\mathcal{E}$ is a tautology, then $\sim \mathcal{E}$ is a contradiction and vice versa.

3. **Definition (valid expression)**
   We say an expression is *valid* if it is a tautology.

   The most straightforward way to determine whether an expression is a tautology or a contradiction is by a complete analysis of all possible values of the expression using its truth table.

   Examples
   Consider the following:

$$\mathcal{E} : P \wedge Q \rightarrow P$$
$$\mathcal{F} : P \wedge \sim P \wedge Q$$

   The corresponding truth tables are

			1	$\mathcal{E}$		*2*	*1*	$\mathcal{F}$
P	Q	$\|\|$	$P \wedge Q$	$\rightarrow P$	$\|\|$	$P \wedge$	$\sim P$	$\wedge Q$
T	T	$\|\|$	T	T	$\|\|$	F	F	F
T	F	$\|\|$	F	T	$\|\|$	F	F	F
F	T	$\|\|$	F	T	$\|\|$	F	T	F
F	F	$\|\|$	F	T	$\|\|$	F	T	F

   The truth values in the column headed $\mathcal{E}$ exhibit the truth values of the expression $\mathcal{E}$; because $\mathcal{E}$ is true in each case we have determined that it is a tautology. In the same way, we have determined that $\mathcal{F}$ is a contradiction.

4. **Definition (implies)**
   An expression $\mathcal{E}$ *implies* an expression $\mathcal{F}$ if $(\mathcal{E} \rightarrow \mathcal{F})$ is a tautology. It is written $\mathcal{E} \Rightarrow \mathcal{F}$.

5. **Definition (equivalent expressions)**
   Expressions $\mathcal{E}$ and $\mathcal{F}$ are called *logically* (or *tautologically*) *equivalent* if $(\mathcal{E} \equiv \mathcal{F})$ is a tautology. It is written $\mathcal{E} = \mathcal{F}$.

   - If $\mathcal{E} = \mathcal{F}$, then $\mathcal{E}$ and $\mathcal{F}$ always have the same truth value for each combination of their component letters.
   - We can establish that if $\mathcal{E} \Rightarrow \mathcal{F}$ and $\mathcal{F} \Rightarrow \mathcal{E}$ then $\mathcal{E}$ and $\mathcal{F}$ are tautologically equivalent, i.e. $\mathcal{E} = \mathcal{F}$.

Examples

Consider the two expressions

$$\mathcal{E} : P \wedge (P \to Q) \to Q$$
$$\mathcal{F} : P \to Q \equiv \sim P \vee Q$$

Their corresponding truth tables are:

		2	*1*	$\mathcal{E}$		*3*	$\mathcal{F}$	*1*	*2*
P	Q	$P \wedge$	$(P \to Q)$	$\to Q$		$P \to Q$	$\equiv$	$\sim P$	$\vee Q$
T	T	T	T	T		T	T	F	T
T	F	F	F	T		F	T	F	F
F	T	F	T	T		T	T	T	T
F	F	F	T	T		T	T	T	T

Because $\mathcal{E}$ is true in each case, we have determined that $\mathcal{E}$ is a tautology or $P \wedge (P \to Q)$ implies Q, i.e.

$$P \wedge (P \to Q) \Rightarrow Q$$

This property is called the *modus ponens* Law.

In the same way, we can determine that $P \to Q$ and $\sim P \vee Q$ are logically equivalent, i.e.

$$P \to Q = \sim P \vee Q.$$

Show

$$\sim (P \vee Q) = \sim P \wedge \sim Q$$

and

$$\sim (P \wedge Q) = \sim P \vee \sim Q.$$

It is sufficient to show that the expressions

$$\mathcal{E} : \sim (P \vee Q) \equiv \sim P \wedge \sim Q$$

and

$$\mathcal{F} : \sim (P \wedge Q) \equiv \sim P \vee \sim Q.$$

are tautologies which we do by constructing their truth tables:

		2	*1*	$\mathcal{E}$	*3*	*5*	*4*
P	Q	$\sim$	$(P \vee Q)$	$\equiv$	$\sim P$	$\wedge$	$\sim Q$
T	T	F	T	T	F	F	F
T	F	F	T	T	F	F	T
F	T	F	T	T	T	F	F
F	F	T	F	T	T	T	T

		2	*1*	$\mathcal{F}$	*3*	*5*	*4*
P	Q	$\sim$	$(P \wedge Q)$	$\equiv$	$\sim P$	$\vee$	$\sim Q$
T	T	F	T	T	F	F	F
T	F	T	F	T	F	T	T
F	T	T	F	T	T	T	F
F	F	T	F	T	T	T	T

The truth values in the columns headed $\mathcal{E}$ and $\mathcal{F}$ are all true, which means that the two pairs of expressions are equivalent.

These properties of the logical connectives $\sim$, $\wedge$ and $\vee$ are known as DeMorgan's Laws.

2.1.7 Additional Connectives

In computer science, in addition to the connectives defined above, the three following connectives are often used.

1. **XOR**
   Recall that the $\vee$ connective is "inclusive" in the sense that $P \vee Q$ is true in the case where both P and Q are true. But, for example, in the statement

   Today is 5 or 6 of June

   the *or* connective should be "exclusive" in the sense that its subsentences cannot both be true.

   - The Exclusive OR of P and Q, denoted by $PxorQ$, is the proposition that is true when exactly one of P and Q is true and false otherwise.
   - The equivalence

     $$PxorQ = (P \vee Q) \wedge \sim (P \wedge Q)$$

     holds for *xor*.

2. **NOR**

   - P *nor* Q is a proposition which is defined by the expression $\sim (P \vee Q)$, i.e.

$$P \; nor \; Q = \sim (P \vee Q).$$

- Also

$$P \; nor \; Q = \sim P \wedge \sim Q \text{ (DeMorgan's law is used).}$$

3. **NAND**

- $P \; nand \; Q$ is a proposition which is defined by the expression $\sim (P \wedge Q)$, i.e.

$$P \; nand \; Q = \sim (P \wedge Q)$$

- Also

$$P \; nand \; Q = \sim P \vee \sim Q \text{ (DeMorgan's law is used).}$$

The truth values of these three connectives are given in the following table:

P	Q	$P \; xor \; Q$	$P \; nor \; Q$	$P \; nand \; Q$
T	T	F	F	F
T	F	T	F	T
F	T	T	F	T
F	F	F	T	T

2.1.8 Logical Equivalences

Up to now we have presented particular logical expressions and introduced a method of establishing their validity. Tautologies and logically equivalent expressions are very useful for making the analytic transformations necessary to simplify complex expressions and to prove other tautologies or establish the equivalence of other expressions. So we now present some of the most important logical equivalences (or logical formulae), where $\mathcal{F}$, $\mathcal{G}$ and $\mathcal{H}$ are arbitrary logical expressions.

- Basic Tautologies
  $\mathcal{F} \vee (\sim \mathcal{F}) = \mathrm{T}$

  $\mathcal{F} \rightarrow \mathcal{F} = \mathrm{T}$

  $\mathcal{F} \equiv \mathcal{F} = \mathrm{T}$

  $(\mathcal{F} \wedge \mathcal{G}) \rightarrow \mathcal{F} = \mathrm{T}$

- Identity Laws
  $\mathcal{F} \wedge \mathrm{T} = \mathcal{F}$

  $\mathcal{F} \vee \mathrm{F} = \mathcal{F}$

- Domination Laws
  $\mathcal{F} \vee \mathrm{T} = \mathrm{T}$

  $\mathcal{F} \wedge \mathrm{F} = \mathrm{F}$

- Idempotent Laws
  $\mathcal{F} \vee \mathcal{F} = \mathcal{F}$

  $\mathcal{F} \wedge \mathcal{F} = \mathcal{F}$

- Double negation Law
  $\sim (\sim \mathcal{F}) \ = \mathcal{F}$

- Commutative Laws
  $\mathcal{F} \vee \mathcal{G} = \mathcal{G} \vee \mathcal{F}$

  $\mathcal{F} \wedge \mathcal{G} = \mathcal{G} \wedge \mathcal{F}$

  $\mathcal{F} \equiv \mathcal{G} = \mathcal{G} \equiv \mathcal{F}$

- Associative Laws
  $(\mathcal{F} \vee \mathcal{G}) \vee \mathcal{H} = \mathcal{F} \vee (\mathcal{G} \vee \mathcal{H})$

  $(\mathcal{F} \wedge \mathcal{G}) \wedge \mathcal{H} = \mathcal{F} \wedge (\mathcal{G} \wedge \mathcal{H})$

  $(\mathcal{F} \equiv \mathcal{G}) \equiv \mathcal{H} \ = \ \mathcal{F} \equiv (\mathcal{G} \equiv \mathcal{H})$

- Distributive Laws
  $\mathcal{F} \vee (\mathcal{G} \wedge \mathcal{H}) = (\mathcal{F} \vee \mathcal{G}) \wedge (\mathcal{F} \vee \mathcal{H})$

  $\mathcal{F} \wedge (\mathcal{G} \vee \mathcal{H}) = (\mathcal{F} \wedge \mathcal{G}) \vee (\mathcal{F} \wedge \mathcal{H})$

- DeMorgan's Laws
  $\sim (\mathcal{F} \wedge \mathcal{G}) = \sim \mathcal{F} \vee \sim \mathcal{G}$

$\sim (\mathcal{F} \vee \mathcal{G}) = \sim \mathcal{F} \wedge \sim \mathcal{G}$

$\sim (\mathcal{F} \rightarrow \mathcal{G}) \ = \ \mathcal{F} \wedge \sim \mathcal{G}$

$\sim (\mathcal{F} \equiv \mathcal{G}) = (\mathcal{F} \equiv \sim \mathcal{G})$

- OR Simplification
  $\mathcal{F} \vee \mathrm{F} = \mathcal{F}$

  $\mathcal{F} \ \vee (\mathcal{F} \wedge \mathcal{G}) = \mathcal{F}$

- AND Simplification
  $\mathcal{F} \wedge \mathrm{T} = \mathcal{F}$

  $\mathcal{F} \wedge (\mathcal{F} \vee \mathcal{G}) = \mathcal{F}$

- Implication Simplification
  $(\mathcal{F} \rightarrow \mathcal{G}) \wedge (\mathcal{F} \rightarrow \mathcal{H}) = (\mathcal{F} \rightarrow \mathcal{G} \wedge \mathcal{H})$

  $(\mathcal{F} \rightarrow \mathcal{H}) \wedge (\mathcal{G} \rightarrow \mathcal{H}) = (\mathcal{F} \vee \mathcal{G} \rightarrow \mathcal{H})$

  $(\mathcal{F} \rightarrow \mathcal{G}) \vee (\mathcal{F} \rightarrow \mathcal{H}) = (\mathcal{F} \rightarrow \mathcal{G} \vee \mathcal{H})$

  $(\mathcal{F} \rightarrow \mathcal{H}) \vee (\mathcal{G} \rightarrow \mathcal{H}) = (\mathcal{F} \wedge \mathcal{G} \rightarrow \mathcal{H})$

- Transitive Laws
  $(\mathcal{F} \rightarrow \mathcal{G}) \wedge (\mathcal{G} \rightarrow \mathcal{H}) \rightarrow (\mathcal{F} \rightarrow \mathcal{H})$

  $(\mathcal{F} \equiv \mathcal{G}) \wedge (\mathcal{G} \equiv \mathcal{H}) \rightarrow (\mathcal{F} \equiv \mathcal{H})$

- Laws of Contradiction
  $\sim (\mathcal{F} \wedge \sim \mathcal{F}) = \mathrm{T}$

  $\mathcal{F} \wedge \sim \mathcal{F} = \mathrm{F}$

- Laws of Contraposition
  $\mathcal{F} \rightarrow \mathcal{G} = \sim \mathcal{G} \rightarrow \sim \mathcal{F}$

  $\mathcal{F} \equiv \mathcal{G} = \sim \mathcal{F} \equiv \sim \mathcal{G}$

- Law of Absorption
  $(\mathcal{F} \to \mathcal{G}) \to (\mathcal{F} \to (\mathcal{F} \wedge \mathcal{G}))$

- Law of Simplification
  $\mathcal{F} \wedge \mathcal{G} \to \mathcal{F}$

- *modus ponens* Law
  $\mathcal{F} \wedge (\mathcal{F} \to \mathcal{G}) \to \mathcal{G}$
- *modus tollens* Law
  $(\mathcal{F} \to \mathcal{G}) \wedge \sim \mathcal{G} \to \sim \mathcal{F}$
- Tautologies for Eliminating Connectives
  $\mathcal{F} \to \mathcal{G} = \sim \mathcal{F} \vee \mathcal{G}$

  $\mathcal{F} \to \mathcal{G} = \sim (\mathcal{F} \wedge \sim \mathcal{G})$

  $\mathcal{F} \vee \mathcal{G} = (\sim \mathcal{F} \to \mathcal{G})$

  $\mathcal{F} \wedge \mathcal{G} = \sim (\mathcal{F} \to \sim \mathcal{G})$

  $\mathcal{F} \equiv \mathcal{G} = (\mathcal{F} \to \mathcal{G}) \wedge (\mathcal{G} \to \mathcal{F})$

  $\mathcal{F} \equiv \mathcal{G} = (\mathcal{F} \wedge \mathcal{G}) \vee (\sim \mathcal{F} \wedge \sim \mathcal{G})$

Multiple disjunction and conjunction
Note that expressions such as

$$\begin{array}{l}((P \vee Q) \vee R) \vee S,\\ P \vee (Q \vee (R \vee S)),\\ P \vee ((Q \vee R) \vee S),\end{array}$$

and so forth are equivalent because of the associative law. For this reason, we will sometimes write any of these expressions without parentheses, as

$$P \vee Q \vee R \vee S$$

In general, we will write a multiple disjunction and a multiple conjunction as the following:

$$\mathcal{F}_1 \vee \mathcal{F}_2 \vee \mathcal{F}_3 \vee \ldots \vee \mathcal{F}_n$$

$$\mathcal{F}_1 \wedge \mathcal{F}_2 \wedge \mathcal{F}_3 \wedge \ldots \wedge \mathcal{F}_n$$

2.1.9 Proof Techniques

1. Use of truth tables
The most straightforward way of proving a given tautology or the equivalence of two given expressions is the use of truth tables and the analysis of all the possible truth values of the expressions. For example, to prove the equivalence

$$\sim (P \equiv Q) = (P \equiv \sim Q)$$

we consider the two expressions:

$$\mathcal{F} : \sim (P \equiv Q)$$

and

$$\mathcal{G} : P \equiv \sim Q$$

and calculate their truth tables

		$\mathcal{F}$	*1*	$\mathcal{G}$	*1*
P	Q	$\sim$	$(P \equiv Q)$	$P \equiv$	$\sim Q$
T	T	F	T	F	T
T	F	T	F	T	F
F	T	T	F	T	T
F	T	F	T	F	F

Then, we analyse the values of these two expressions. We determine that the given equivalence is true because the truth values in the columns headed $\mathcal{F}$ and $\mathcal{G}$ are the same in each case.

However, the use of truth tables is tedious and unmanageable if the expressions involved are long and complicated, in particular if they contain many different propositional letters, because the size of the table increases exponentially with this number (for example, the truth table has $2^5 = 32$ rows for an expression containing 5 propositional letters).

So, we need some analytic methods of proving logical properties of propositions.

2. Substitution method
Analytic transformations are often used for proving tautologies and equivalences. Such analytic

transformations must not add anything to the meaning of an expression.

Let us introduce some notion for the operation of replacing subexpressions of a given expression with other subexpressions.

Definition (substitution)
Let $\mathcal{G}$ and $\mathcal{H}$ be expressions.

If $\mathcal{F}[\mathcal{G}]$ is an expression, which may or may not contain an occurrence of the expression $\mathcal{G}$ as a subexpression, then $\mathcal{F}[\mathcal{H}]$ denotes any of the expressions obtained by replacing zero, one, or more occurrences of $\mathcal{G}$ in $\mathcal{F}[\mathcal{G}]$ with the expression $\mathcal{H}$.

For example, if

$$\mathcal{F}[P] : (P \vee Q),$$

then $\mathcal{F}[Q]$ denotes either of the expressions

$$(P \vee Q) \text{ or } (Q \vee Q).$$

Note that the substitution does not necessarily denote a unique expression.

For example, if the expression $\mathcal{F}[P] : P \vee P$ is given, then $\mathcal{F}[Q]$ may denote any of the following expressions:

$P \vee P$	(replacing zero occurrences of P)
$Q \vee P$	(replacing the first occurrence of P)
$P \vee Q$	(replacing the second occurrence of P)
$Q \vee Q$	(replacing both occurrences of P)

Thus the substitution above represents any of four expressions. So, if we wish to specify which occurrences are to be replaced, we must do so in words.

Theorem 1: For any expressions $\mathcal{G}$, $\mathcal{H}$ and $\mathcal{F}[\mathcal{G}]$, if $\mathcal{G}$ and $\mathcal{H}$ are equivalent, then $\mathcal{F}[\mathcal{G}]$ and $\mathcal{F}[\mathcal{H}]$ are equivalent, i.e.

$$\mathcal{G} = \mathcal{H} \Rightarrow \mathcal{F}[\mathcal{G}] = \mathcal{F}[\mathcal{H}]$$

This proposition is intuitively clear because $\mathcal{G}$ and $\mathcal{H}$ have the same truth value under any combination of truth values of their components. Therefore in determining the truth values of $\mathcal{F}[\mathcal{G}]$ and $\mathcal{F}[\mathcal{H}]$ under these combinations, we obtain the same result in each case.

Thus if two expressions are equivalent, we may replace any occurrence of one of them with the other, obtaining an equivalent expression.

Corollary: For any expressions $\mathcal{G}$, $\mathcal{H}$ and $\mathcal{F}[\mathcal{G}]$, if $\mathcal{G}$ and $\mathcal{H}$ are equivalent and $\mathcal{F}[\mathcal{G}]$ is a tautology, then $\mathcal{F}[\mathcal{H}]$ is also a tautology.

Chains of Equivalences

According to the Transitive Laws (see Section 2.1.8) the equivalence connective is transitive, i.e. the expression

$$(\mathcal{F} \equiv \mathcal{G}) \wedge (\mathcal{G} \equiv \mathcal{H}) \rightarrow (\mathcal{F} \equiv \mathcal{H})$$

is a tautology. In addition, the equivalence relationship between expressions is transitive, that is,

$$\text{If } \mathcal{F} = \mathcal{G} \text{ and } \mathcal{G} = \mathcal{H}, \text{ then } \mathcal{F} = \mathcal{H}.$$

This provides another way of proving the validity of certain expressions.

Suppose we would like to prove the validity of an expression $\mathcal{F}$. We attempt to find a sequence of expressions $\mathcal{F}_1$, $\mathcal{F}_2$, $\ldots$ $\mathcal{F}_n$ such that

$$\begin{aligned} \mathcal{F} &= \mathcal{F}_1 \\ \mathcal{F}_1 &= \mathcal{F}_2 \\ &\vdots \\ \mathcal{F}_{n-1} &= \mathcal{F}_n \end{aligned}$$

where $\mathcal{F}_n$ is known to be a tautology.

Then we can conclude that $\mathcal{F}$ is a tautology.

We write this chain of equivalences in the form:

$$\mathcal{F} = \mathcal{F}_1 = \mathcal{F}_2 = \cdots = \mathcal{F}_{n-1} = \mathcal{F}_n$$

Examples

- Prove that the expression

$$\sim (\sim \mathcal{F}) \rightarrow \mathcal{F} \wedge (\mathcal{F} \vee \mathcal{G})$$

is a tautology. We may write the following chain:

	Expressions	Rules used
	$\sim (\sim \mathcal{F}) \rightarrow \mathcal{F} \wedge (\mathcal{F} \vee \mathcal{G})$	
=	$\mathcal{F} \rightarrow \mathcal{F} \wedge (\mathcal{F} \vee \mathcal{G})$	Double negation
=	$\mathcal{F} \rightarrow \mathcal{F}$	AND - simplification
=	T	Basic tautologies

- Prove that the expressions

$$(\sim \mathcal{F} \rightarrow \mathcal{G}) \text{ and } (\sim \mathcal{G} \rightarrow \mathcal{F})$$

are equivalent.
We may write the following chain:

	Expressions	Rules used
	$\sim \mathcal{F} \rightarrow \mathcal{G}$	
=	$\sim (\sim \mathcal{F}) \vee \mathcal{G}$	Elimination of $\rightarrow$
=	$\mathcal{F} \vee \mathcal{G}$	Double negation
=	$\mathcal{G} \vee \mathcal{F}$	Commutative Law
=	$\sim (\sim \mathcal{G}) \vee \mathcal{F}$	Double negation
=	$(\sim \mathcal{G} \rightarrow \mathcal{F})$	Elimination of $\rightarrow$ (*reverse*)

3. Normal Forms

Often it is beneficial to reduce a formula to its simplest form, where only the connectives $\wedge$ and $\vee$ and negation are applied to elementary propositions. There are two such forms in propositional logic.

Definition (disjunctive terms)

- Any propositional letters and their negations are disjunctive terms.
- If $\mathcal{F}$ and $\mathcal{G}$ are disjunctive terms, then so is the expression $(\mathcal{F} \vee \mathcal{G})$.

Examples

$$P,\ Q,\ \sim P,\ (\sim P \vee Q),\ ((\sim P \vee Q) \vee R),\ \text{and}\ (\sim P \vee Q \vee R) \vee (R \vee \sim P)$$

are disjunctive terms.

Definition (conjunctive normal form)
The conjunctive normal form (CNF) is a conjunction of disjunctive terms.

So, in a conjunctive normal form, the major connectives are conjunctions.

Examples
The expressions

$$P \wedge Q,\ \sim P \wedge (\sim P \vee Q),\ \text{and}\ (\sim P \vee Q \vee R) \wedge (R \vee \sim P) \wedge (P \vee Q)$$

are in CNF.

Transformation to CNF

Any given logical expression can be transformed to CNF using the following algorithm:

1. Write the given expression.
2. Replace all subexpressions of the form $\mathcal{F} \equiv \mathcal{G}$ with $(\mathcal{F} \to \mathcal{G}) \wedge (\mathcal{G} \to \mathcal{F})$.
3. Replace all subexpressions of the form $\mathcal{F} \to \mathcal{G}$ with $(\sim \mathcal{F} \vee \mathcal{G})$.
4. Replace
   (a) all subexpressions of the form $\sim (\mathcal{F} \wedge \mathcal{G})$ with $\sim \mathcal{F} \vee \sim \mathcal{G}$,
   (b) all subexpressions of the form $\sim (\mathcal{F} \vee \mathcal{G})$ with $\sim \mathcal{F} \wedge \sim \mathcal{G}$,
   (c) all subexpressions of the form $\sim (\sim \mathcal{F})$ with $\mathcal{F}$.
5. Repeat step 4 until there are no negations of subexpressions, only of propositional letters.
6. Replace
   (a) all subexpressions of the form $\mathcal{H} \vee (\mathcal{F} \wedge \mathcal{G})$ with $(\mathcal{H} \vee \mathcal{F}) \wedge (\mathcal{H} \vee \mathcal{G})$,

(b) all subexpressions of the form $(\mathcal{F} \wedge \mathcal{G}) \vee \mathcal{H}$ with $(\mathcal{F} \vee \mathcal{H}) \wedge (\mathcal{G} \vee \mathcal{H})$.

7. Repeat step 6 until the expression is in CNF.
8. The result is the conjunctive normal form of the given expression.

Examples

- Given the expression $(P \to Q) \wedge (P \to R)$,
  its transformation to CNF is:

$$(P \to Q) \wedge (P \to R) \quad = \quad (\sim P \vee Q) \wedge (\sim P \vee R)$$

- the expression $\sim (P \to Q) \vee (P \to R)$
  is transformed to CNF via:

$$\begin{aligned} \sim (P \to R) \vee (Q \to R) &= \sim (\sim P \vee R) \vee (\sim Q \vee R) \\ &= (\sim (\sim P) \wedge \sim R) \vee (\sim Q \vee R) \\ &= (P \wedge \sim R) \vee (\sim Q \vee R) \\ &= (P \vee (\sim Q \vee R)) \ \wedge \ (\sim R \vee (\sim Q \vee R)) \\ &= (P \vee \sim Q \vee R) \wedge (\sim R \vee \sim Q \vee R) \end{aligned}$$

Definition (disjunctive normal form)
Conjunctive terms can be defined similarly to disjunctive ones. Then the disjunctive normal form (DNF) is a disjunction of conjunctive terms.

So, in a disjunctive normal form, the major connectives are disjunctions.

Examples
The expressions

$$P \vee Q, \sim P \vee (\sim P \wedge Q), \text{ and } (\sim P \wedge Q \wedge R) \vee (R \wedge \sim P) \vee (P \wedge Q)$$

are in DNF.

Transformation to DNF

Any given logical expression can be transformed to DNF using the following algorithm:

1. Write the given expression.
2. Replace all subexpressions of the form $\mathcal{F} \equiv \mathcal{G}$ with $(\mathcal{F} \to \mathcal{G}) \wedge (\mathcal{G} \to \mathcal{F})$.
3. Replace all subexpressions of the form $\mathcal{F} \to \mathcal{G}$ with $(\sim \mathcal{F} \vee \mathcal{G})$.
4. Replace
   (a) all subexpressions of the form $\sim (\mathcal{F} \wedge \mathcal{G})$ with $\sim \mathcal{F} \vee \sim \mathcal{G}$,
   (b) all subexpressions of the form $\sim (\mathcal{F} \vee \mathcal{G})$ with $\sim \mathcal{F} \wedge \sim \mathcal{G}$,
   (c) all subexpressions of the form $\sim (\sim \mathcal{F})$ with $\mathcal{F}$.
5. Repeat step 4 until there are no negations of subexpressions, only of propositional letters.
6. Replace
   (a) all subexpressions of the form $\mathcal{H} \wedge (\mathcal{F} \vee \mathcal{G})$ with $(\mathcal{H} \wedge \mathcal{F}) \vee (\mathcal{H} \wedge \mathcal{G})$,
   (b) all subexpressions of the form $(\mathcal{F} \vee \mathcal{G}) \wedge \mathcal{H}$ with $(\mathcal{F} \wedge \mathcal{H}) \vee (\mathcal{G} \wedge \mathcal{H})$.
7. Repeat step 6 until the expression is in DNF.
8. The result is the disjunctive normal form of the given expression.

Examples

- Given expression $(P \to Q) \wedge (P \to R)$,
  its transformation to DNF is:

$$\begin{array}{rl} & (P \to Q) \wedge (P \to R) \\ = & (\sim P \vee Q) \wedge (\sim P \vee R) \\ = & ((\sim P \vee Q) \wedge \sim P) \vee ((\sim P \vee Q) \wedge R) \\ = & (\sim P \wedge \sim P) \vee (Q \wedge \sim P) \vee (\sim P \wedge R) \vee (Q \wedge R) \\ = & \sim P \vee (Q \wedge \sim P) \vee (\sim P \wedge R) \vee (Q \wedge R) \end{array}$$

  Also we can get a simple form of the given expression directly using the distributive law:

$$\begin{array}{rcl} (P \to Q) \wedge (P \to R) & = & (\sim P \vee Q) \wedge (\sim P \vee R) \\ & = & \sim P \vee (Q \wedge R) \end{array}$$

- Consider the expression $\sim (P \to Q) \vee (P \to R)$.
  Its transformation to DNF is:

$$\begin{array}{rcl} \sim (P \to R) \vee (Q \to R) & = & \sim (\sim P \vee R) \vee (\sim Q \vee R) \\ & = & (\sim (\sim P) \wedge \sim R) \vee (\sim Q \vee R) \\ & = & (P \wedge \sim R) \vee (\sim Q \vee R) \\ & = & (P \wedge \sim R) \vee \sim Q \vee R \end{array}$$

These normal forms provide a mechanism for checking whether a given formula is a tautology or whether two expressions are equivalent.

When an expression is in disjunctive normal form then if two of its terms taken together form a tautology (e.g. $P \vee \sim P$) then the entire disjunction must be a tautology. Also when an expression is in conjunctive normal form then if two of its terms taken together form a contradiction (e.g. $P \wedge \sim P$) then the entire conjunction must be a contradiction.

Thus, we may use one of the following strategies:

1. One way to establish the validity of an expression is first transform it to its disjunctive normal form and then reduce it to a known tautology.
2. Another way is to negate the expression, transform it to its conjunctive normal form and then reduce it to a known contradiction - if its negation is a contradiction, then the original expression must be a tautology.
3. To establish equivalence of two expressions, first transform them both to CNF or DNF, and then compare the results. If the results are the same the two expressions are equivalent.

Examples

- Check the equivalence of expressions $\mathcal{F} : \sim ((P \to \sim Q) \wedge (R \to P))$ and $\mathcal{G} : (Q \to \sim P) \to \sim (R \to P)$

$$\begin{aligned} \mathcal{F} : \sim ((P \to \sim Q) \wedge (R \to P)) &= \sim ((\sim P \vee \sim Q) \wedge (\sim R \vee P)) \\ &= \sim (\sim P \vee \sim Q) \vee \sim (\sim R \vee P) \\ &= (P \wedge Q) \vee (R \wedge \sim P) \end{aligned}$$

$$\begin{aligned} \mathcal{G} : (Q \to \sim P) \to \sim (R \to P) &= \sim (Q \to \sim P) \vee \sim (R \to P) \\ &= \sim (\sim Q \vee \sim P) \vee \sim (\sim R \vee P) \\ &= (Q \wedge P) \vee (R \wedge \sim P) \\ &= (P \wedge Q) \vee (R \wedge \sim P) \end{aligned}$$

  Therefore, $\mathcal{F}$ and $\mathcal{G}$ are equivalent.

- Prove the validity of the expression

$$(P \to Q) \to (P \to (P \wedge Q))$$

  We transform the given expression to DNF:

$$
\begin{array}{lll}
 & (P \to Q) \to (P \to (P \wedge Q)) & \\
= & \sim (P \to Q) \vee (P \to (P \wedge Q)) & \\
= & \sim (\sim P \vee Q) \vee (\sim P \vee (P \wedge Q)) & \\
= & (P \wedge \sim Q) \vee (\sim P \vee (P \wedge Q)) & \\
= & (P \wedge \sim Q) \vee \sim P \vee (P \wedge Q) & \textit{DNF} \\
= & (P \wedge \sim Q) \vee (P \wedge Q) \vee \sim P & \textit{Commutative law} \\
= & P \wedge (\sim Q \vee Q) \vee \sim P & \textit{Distributive law} \\
= & P \wedge \mathrm{T} \vee \sim P & \textit{Basic tautology} \\
= & P \vee \sim P & \textit{AND simplification} \\
= & \mathrm{T} & \textit{Basic tautology}
\end{array}
$$

2.2 Predicate Logic

2.2.1 Introduction

From our study of propositional logic we can determine that a sentence such as

All people are mortal or All people are not mortal

is true, because it is an instance of the valid propositional logic expression

$$P \vee (\sim P)$$

with P as the proposition 'All people are mortal'.

There are some sentences, however, that we cannot tell to be true or false simply by their form because they are not instances of any valid expressions in propositional logic. Consider for example the following sentences:

There is an even prime number and all prime numbers are not even

and

All men are mortal and *Socrates is a man,*
therefore Socrates is mortal

We can see immediately that both sentences are true, but the language and methods of propositional logic are not enough to express them fully or to prove their validity. For example, in the

first sentence we might take P to be the proposition 'There is an even prime number' and Q to be the proposition 'All prime numbers are even', and we could then write the entire sentence in the form:

$$P \vee \sim Q$$

but we cannot deduce that this is true because it is of course not a tautology.

The reason we cannot establish the validity of this kind of sentence is that the language of propositional logic is too primitive to express properties of an object (such as being a prime number or a man) or relationships between objects.

Sentences involving variables, such as '$x = y + 3$' or '$x > 3$' are often found in mathematical assertions and in computer programs. We cannot determine whether such sentences are true or false if the values of their variables are unknown, which means that we cannot in general determine whether combinations of them, such as $(x \leq -2) \vee (2 \leq x)$, are true or false using the methods of propositional logic.

The language and methods of predicate logic extend propositional logic by enabling us to speak about objects, their properties, and the relationships between them, and thus provide a way of reasoning about this type of sentence.

Definition (symbols)
The sentences of predicate logic are made up of the following symbols:

- The truth constants
  true and *false*
- The propositional letters
  $P, Q, R, S, P_1, Q_1, R_1, S_1, P_2, ...$ (the capital letters, possibly with a numerical subscript)
- The constants
  $a, b, c, a_1, b_1, c_1, a_2, b_2, c_2, ...$
- The variables
  $x, y, z, u, x_1, y_1, z_1, u_1, x_2, y_2, ...$
- The predicate symbols
  $p, q, r, p_1, q_1, r_1, p_2, q_2, ...$

Intuitively, the constants and variables denote objects, and predicate symbols denote relations between these objects. Note that the propositional letters of propositional logic are part of the

language of predicate logic.

Definition (unary predicates)

Let **D** be a set and x be a variable which takes some value from the set **D**. Then, a unary predicate defined on the set **D** is any assertive sentence involving the variable x, and is denoted by $p(x)$, $q(x)$, $r(x)$, etc. The set **D** is called the domain of the variable x.

Examples

1. Let **D** be the set of all people. Then
   $p(y)$: 'y is mortal'
   and
   $q(y)$: 'y is wise'
   are predicates on **D** .
2. Let **R** be the set of all real numbers. Then
   $p_1(x)$: '$x \leq 3$'
   and
   $q_1(x)$: '$(x \leq -2) \vee (2 \leq x)$'
   are predicates on **R**.
3. Let **N** be the set of all natural numbers. Then
   $p_2(n)$: 'n is an even number'
   and
   $q_2(n)$: '$2^{2^n} + 1$ is a prime number'
   are predicates on **N**.

Predicates themselves are not propositions because they are neither true nor false. They instead define properties of their subjects, which are represented abstractly by the variables they contain. For example, in the predicate $p(y)$: 'y is mortal', the variable y is the subject and '*is mortal*' is its property. The value of a predicate depends on the values of its variables, and when specific values are substituted for these variables the predicate evaluates to either true or false, i.e. it becomes a proposition.

Examples
For the predicates defined above we have:

- $p_1(\pi)$: '$\pi \leq 3$' = F and
  $q_1(\pi)$: '$(\pi \leq -2) \vee (2 \leq \pi)$' = T
  but $p_1(1)$: '$1 \leq 3$' = T.

- $p_2(12)$: '12 is an even number' = T and
  $q_2(2)$: '$2^{2^2}+1$ is a prime number' =T, because $2^{2^2}+1 = 2^4+1 = 16+1 = 17$ is a prime number.

Definition (binary predicate)

Let **D** and **L** be sets and x and y be variables which take their values from the sets **D** and **L** respectively. Then a binary predicate defined on the sets **D** and **L** is any assertive sentence involving the variables x and y, and is denoted by $p(x,y)$, $q(x,y)$, $r(x,y)$, etc. The sets **D** and **L** are called the domains of the variables x and y respectively.

Examples

1. Let x and y be variables which take values from **R** (the set of all real numbers). Then
   $p(x,y)$: '$x \leq y$'
   and
   $q(x,y)$: '$x^2+y^2 \geq 4$'
   are binary predicates defined on **R**.
2. Let **D** be the set of all people and x and y be variables taking values from **D**. Then
   $p_1(x,y)$: 'x and y are roommates'
   and
   $q_1(x,y)$: 'x is taller than y'
   are binary predicates defined on **D**.

Definition (predicate)
A predicate is an assertive sentence containing a specific number of variables. It becomes a proposition when specific values taken from the domains of the variables are substituted in place of the variables.

A predicate containing n ($n \geq 1$) variables is also called an n-ary predicate. 1-ary (unary) and 2-ary (binary) predicates have been defined specifically above.

2.2.2 Truth sets

A set can be defined by simply enclosing the set elements in braces. For instance, the set $\mathbf{S} = \{2,3,5,7\}$ is the set of prime numbers less than 10. When using symbols, sets are typically represented by upper case letters and set elements by lower case. A particular number x may or may not belong to, or be a member or an element of, the set **S**. We use the predicate '$x \in \mathbf{S}$'

to mean that x is an element of the set **S**, and the predicate '$x \notin \mathbf{S}$' to indicate the converse, i.e. that x is not an element of (or x is not in) **S**.

Definition (Venn diagrams)
Venn diagrams are a way of representing sets pictorially. Circles represent sets, a rectangle around the circles represents the Universal set, and shading is used to illustrate the elements of the sets.

The diagrams are not a complete representation of sets or their relationship, e.g. the size of sets and their individual elements cannot be represented.

We have mentioned that when values (or elements) are substituted for variables in a predicate it becomes a proposition, i.e. its value is either true or false. All the elements that make the predicate a true proposition form one set and the elements which make the predicate a false proposition form another set.

Definition (truth set)
Let $p(x)$ be a predicate on a set **D**. Then the *truth set* of the predicate $p(x)$ is the set of all elements x of the domain **D** which make $p(x)$ true.

We denote the truth set of $p(x)$ by **P**. This is written symbolically as:

$$\mathbf{P} = \{x \in \mathbf{D} \mid p(x)\}$$

which is read as "**P** is the set of all x in **D** such that $p(x)$". (the vertical bar '|' is read as "such that").

2.2.3 Connections of Predicates

We have defined logical operators $\sim$, $\wedge$, $\vee$, $\rightarrow$, and $\equiv$ for propositions. Now we define these operators for predicates.

Definitions
Let $p(x)$ and $q(x)$ be predicates defined on the same domain **D**.

1. **NOT**

The negation of $p(x)$ is the predicate defined on $\mathbf{D}$ which is true only for those elements x of $\mathbf{D}$ for which $p(x)$ is false. It is denoted by $\sim p(x)$ and its truth set is denoted by $\overline{\mathbf{P}}$.

The domain $\mathbf{D}$ is thus divided into two parts by the predicate $p(x)$: $\mathbf{P}$ and $\overline{\mathbf{P}}$. These sets are shown in the Venn diagrams in Figure 1.

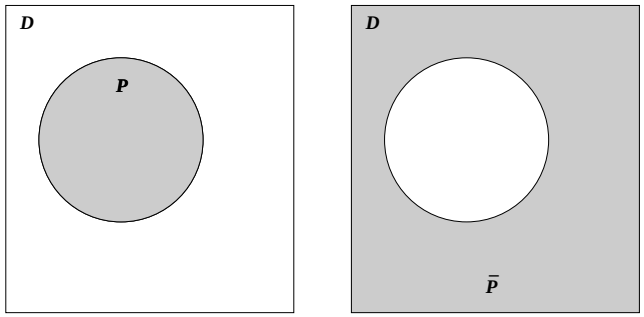

Figure 1: The truth sets of predicates $p(x)$ and $\sim p(x)$

2. **AND**
   The conjunction $p(x) \wedge q(x)$ is the predicate defined on $\mathbf{D}$ which is true only for those elements x of $\mathbf{D}$ for which both $p(x)$ and $q(x)$ are true.

3. **OR**
   The disjunction $p(x) \vee q(x)$ is the predicate defined on $\mathbf{D}$ which is true for those elements x of $\mathbf{D}$ for which either $p(x)$ or $q(x)$ or both are true. The truth sets of the disjunction and conjunction of two predicates are shown in Figure 2.

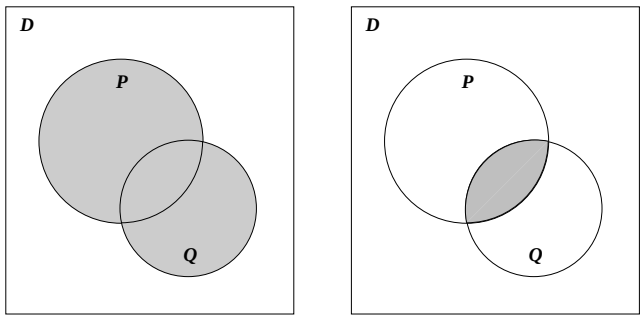

Figure 2: The truth sets of predicates $p(x) \vee q(x)$ and $p(x) \wedge q(x)$

4. **CONDITIONAL**
   The conditional $p(x) \rightarrow q(x)$ is the predicate defined on $\mathbf{D}$ which is false only for those elements x of $\mathbf{D}$ for which $p(x)$ is true and $q(x)$ is false. It is equivalent to the predicate $\sim p(x) \vee q(x)$.

5. **EQUIVALENCE**
   The equivalence $p(x) \equiv q(x)$ is the predicate defined on **D** which is true for all those elements x of **D** for which $p(x)$ and $q(x)$ have the same truth value.
   The truth sets of the disjunction and conjunction of two predicates are shown in Figure 3.

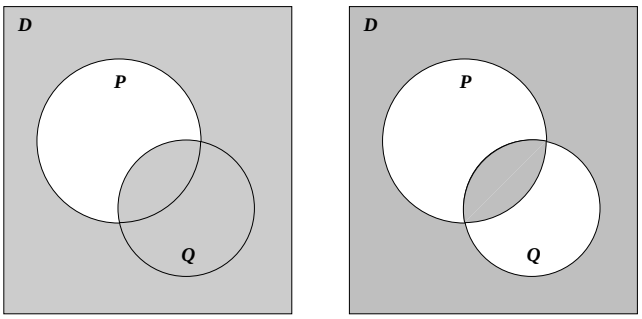

Figure 3: The truth sets of predicates $p(x) \rightarrow q(x)$ and $p(x) \equiv q(x)$

Although we have defined these connectives specifically for unary predicates, the definitions apply with obvious extension to arbitrary predicates.

2.2.4 Quantifiers

We are often interested in statements which indicate the number of elements for which a particular predicate (a property) is true. In particular, if $p(x)$ is a predicate on **D**, we often deal with the following statements:

- $p(x)$ is true for all values of x,
- $p(x)$ is true for at least one value of x,
- $p(x)$ is true for exactly one value of x.

In logic, we write these statements as $\forall x p(x)$, $\exists x p(x)$ and $\exists! x p(x)$ respectively.

The symbol $\forall$ is translated as "for all", "for each", or "for every", and is known as the *universal quantifier sign*. The symbol $\exists$ is the *existential quantifier sign*, and means variously "for some", "there exists", or "for at least one". The symbol $\exists!$ is translated as "there exists exactly one". The string $\forall x$ is called *universal quantification*, and $\exists x$ and $\exists! x$ are called *existential quantification* and *unique existential quantification* respectively. The predicate $p(x)$ appearing in such statements is said to be the *scope* of the corresponding quantifier.

Predicates, as mentioned above, are not propositions. However the quantifiers change predicates into propositions. This process is called quantification.

Definition (universal statements)
A universal statement containing a predicate is a proposition that is true if, and only if, the predicate is true for every value of its variable within the given domain.

Definition (existential statements)
An existential statement of the form $\exists x p(x)$ containing a predicate is a proposition that is true if there is at least one value within the variable's domain for which the predicate is true. An existential statement of the form $\exists! x p(x)$ containing a predicate is a proposition that is true if there is exactly one value within the variable's domain for which the predicate is true.

Examples

- Let $\mathbf{B}$ be the set of all species of non-extinct birds, and x be a variable such that $x \in \mathbf{B}$. Let $q(x)$ be the predicate 'x can fly'.

  The universal statement of this predicate can then be represented as
  $\forall x\, q(x)$ which corresponds to the statement that all species of non-extinct birds can fly.

  It is obvious that $\forall x\, q(x)$ is false since there exist species of birds for which the predicate is false (ostriches and penguins, for example, are flightless). Thus $\exists x \sim q(x)$ is true.

- Let $n \in \mathbf{N}$ and $p(n)$: '$2^{2^n}+1$ is a prime number'. Check whether the statement $\forall n\, p(n)$ is true.

  We can easily determine that $p(1)$, $p(2)$ and $p(3)$ are true because the numbers 5 ($2^{2^1}+1 = 2^2+1 = 5$), 17 ($2^{2^2}+1 = 2^4+1 = 16+1 = 17$), and 257 ($2^{2^3}+1 = 2^8+1 = 256+1 = 257$) are prime. But from this we cannot say that $\forall n\, p(n)$ is true because we have not checked all natural numbers. Checking all numbers in this way is impossible because the set $\mathbf{N}$ is infinite. However, the 17^{th} century mathematician Euler found that $p(5)$ is false ($2^{2^5}+1 = 2^{32}+1 = 4294967296+1 = 4294967297$ is not prime). This one case is sufficient to prove that $\forall n\, p(n)$ is false, i.e. there exists a number m for which $p(m)$ is not true so $p(n)$ cannot be true for all n. However, we have determined that the predicate $\exists n p(n)$ is true by our evaluation of $p(1)$, for example.

- Let $x \in \mathbf{Z}$ (the set of all integers) and let
  $p(x)$: '$2x^2 - 8x + 7 \leq 0$'. Check whether the statement $\exists x p(x)$ is true.

  We can rewrite the given predicate in the following equivalent forms:
  $p(x)$: '$2x^2 - 8x + 8 - 1 \leq 0$' or

$p(x)$: '$2(x^2 - 4x + 4) - 1 \leq 0$' or
$p(x)$: '$2(x-2)^2 - 1 \leq 0$'.

From the last, we can easily see that $p(2)$ is true (because $-1 \leq 0$), which means that there exists an integer for which $p(x)$ is true. Therefore, $\exists x p(x)$ is true.

- Let $n \in \mathbf{N}$, and let
  $p(n)$: 'n is a prime number' and $q(n)$: 'n is an even number'.
  Then $p(n) \wedge q(n)$ means that 'n is an even prime number'.

  Therefore, $\exists! n(p(n) \wedge q(n))$ and also $\exists n(p(n) \wedge q(n))$ are true because 2 is the only even prime number, but $\forall n(p(n) \wedge q(n))$ is false because 5 and 7 are odd prime numbers.

2.2.5 Quantification over empty domains

Let $p(x)$ be a predicate defined on a domain $\mathbf{D}$ which is empty, i.e. $\mathbf{D} = \emptyset$.

Consider the statement $\exists x p(x)$.
$\exists x p(x)$ is equivalent to the statement 'there is some element x such that $x \in \mathbf{D} \wedge p(x)$' which is false because $x \in \mathbf{D}$ is false when $\mathbf{D} = \emptyset$. That is, we arrive at

$\exists x p(x) =$ *false* when the domain of the predicate is empty.

Now consider the statement $\forall x p(x)$.
$\forall x p(x)$ is equivalent to the statement 'for every element x, if x in $\mathbf{D}$ then $p(x)$ is true' which can be written symbolically as '$x \in \mathbf{D} \rightarrow p(x)$'. The last statement is equivalent to '*false* $\rightarrow p(x)$' which is automatically true irrespective of the value of $p(x)$. That is, we arrive at

$\forall x p(x) =$ *true* when the domain of the predicate is empty.

2.2.6 Negation of quantified statements

As stated above, any quantified unary statement is a proposition, so it can be negated.

Let $p(x)$ be a predicate defined on $\mathbf{D}$.

Negating universal statements
The negation of the universal statement $\forall x p(x)$ can be formed in two ways:

- The entire statement is negated, i.e. $\sim (\forall x p(x))$, which means '$p(x)$ is not true for all x'.
- The statement '$p(x)$ is not true for all x' is logically equivalent to the statement 'for at least one x, $p(x)$ is not true'. This is equivalent to $\exists x(\sim p(x))$.

These two different form of negation must be equivalent to each other, so we obtain the formula:

$$\sim (\forall x p(x)) = \exists x(\sim p(x)).$$

Negating existential statements
Similarly, the negation of the existential statement $\exists x p(x)$ can be formed in two ways:

- The entire statement is negated, i.e. $\sim (\exists x p(x))$, which means 'there does not exist x for which $p(x)$ is true'.
- The statement 'there does not exist x for which $p(x)$ is true' is logically equivalent to the statement 'for all x, $p(x)$ is not true'. This is equivalent to $\forall x(\sim p(x))$.

Again, the two forms must be equivalent, so we obtain the formula:

$$\sim (\exists x p(x)) = \forall x(\sim p(x)).$$

The formulae

$$\sim (\forall x p(x)) = \exists x(\sim p(x)).$$

and

$$\sim (\exists x p(x)) = \forall x(\sim p(x)).$$

are DeMorgan's laws for quantified statements.

2.2.7 Multiple quantifiers

Free and bound variables
Predicates contain variables which can be substituted by any values of their domain. Quantified statements, for example $\forall x p(x)$ or $\exists x p(x)$, also contain variables, but these variables are not real variables in the sense that they cannot be substituted by values.

If a variable v is in the scope of a quantifier, $\forall v$ or $\exists v$, then it is said to be a bound variable or bound by the quantifier. On the other hand, a variable which is not in the scope of any quantifier is said to be free.

Example
In the statement

$$\forall x(p(x,y) \wedge q(x))$$

the variable x is bound and y is a free variable.

This example illustrates that when one of the variables in an n-ary ($n \geq 2$) predicate is quantified, we obtain a new $n-1$ -ary predicate. Therefore, we can quantify any of the variables in this $n-1$ -ary predicate, i.e. we can have more that one quantifier in a statement.

Examples

- Let $p(x,y)$ be a predicate, where $x \in D_1$ and $y \in D_2$. Then we can construct unary predicates by quantifying one of the variables of $p(x,y)$ for example:

Predicates	Domain
$q(x) : \forall y\, p(x,y)$	D_1
$r(x) : \exists y\, p(x,y)$	D_1
$s(y) : \forall x\, p(x,y)$	D_2
$t(y) : \exists x\, p(x,y)$	D_2

  Therefore, we can write the following statements:

$$\forall x q(x) = \forall x \forall y\, p(x,y) \qquad \exists x q(x) = \exists x \forall y\, p(x,y)$$

$$\forall x r(x) = \forall x \exists y\, p(x,y) \qquad \exists x r(x) = \exists x \exists y\, p(x,y)$$

$$\forall y s(y) = \forall y \forall x\, p(x,y) \qquad \exists y s(y) = \exists \forall x\, p(x,y)$$

$$\forall y t(y) = \forall y \exists x\, p(x,y) \qquad \exists y t(y) = \exists y \exists x\, p(x,y)$$

- Let $\mathbf{D} = \{a, b, c, d, e\}$, $x, y \in \mathbf{D}$ and $p(x,y)$ be a predicate defined by the table

$x \backslash y$	a	b	c	d	e
a	T	T	F	F	F
b	F	F	F	F	F
c	F	T	T	F	T
d	F	T	T	T	F
e	F	T	F	T	T

Then all the predicates

$$q(x) : \forall y\, p(x,y),$$
$$r(x) : \exists y\, p(x,y),$$
$$s(y) : \forall x\, p(x,y),$$
$$t(y) : \exists x\, p(x,y)$$

are defined on $\mathbf{D}$ and have the following values:

x	a	b	c	d	e
$q(x)$	F	F	F	F	F
$r(x)$	T	F	T	T	T

y	a	b	c	d	e
$s(y)$	F	F	F	F	F
$t(y)$	T	T	T	T	T

Therefore, we establish the following propositions:

$$\forall x q(x) = \forall x \forall y\, p(x,y) = \mathrm{F} \qquad \exists x q(x) = \exists x \forall y\, p(x,y) = \mathrm{F}$$

$$\forall x r(x) = \forall x \exists y\, p(x,y) = \mathrm{F} \qquad \exists x r(x) = \exists x \exists y\, p(x,y) = \mathrm{T}$$

$$\forall y s(y) = \forall y \forall x\, p(x,y) = \mathrm{F} \qquad \exists y s(y) = \exists y \forall x\, p(x,y) = \mathrm{F}$$

$$\forall y t(y) = \forall y \exists x\, p(x,y) = \mathrm{T} \qquad \exists y t(y) = \exists y \exists x\, p(x,y) = \mathrm{T}$$

Observing these propositions shows that

$$\forall x \forall y\, p(x,y) = \forall y \forall x\, p(x,y),$$
$$\exists x \exists y\, p(x,y) = \exists y \exists x\, p(x,y),$$

but

$$\exists x \forall y \, p(x,y) \;\neq\; \forall y \exists x \, p(x,y)$$

In general, it turns out that the order of the quantifiers of different variables does not influence the value of a statement with multiple quantifiers if all the quantifiers are the same. But the value of a statement containing different quantifiers depends on the order of the quantifiers.

Negating statements with multiple quantifiers

Recall that, according to DeMorgan's laws, the negation of a 'for all' statement is a 'there exists' statement, and vice versa, i.e.

$$\sim (\forall x \, p(x)) = \exists x (\sim p(x)).$$

and

$$\sim (\exists x \, p(x)) = \forall x (\sim p(x)).$$

Using these formulae we derive the negation of statements with multiple quantifiers. Consider, for example, the statement

$$\sim \forall x \exists y \, p(x,y)$$

We first break it down into its logical component parts:

$$\sim \forall x \, (\exists y \, p(x,y))$$

Now we negate the outer quantifier using DeMorgan's laws:

$$\exists x \sim (\exists y \, p(x,y))$$

Applying the same operation to the inner quantified statement then yields:

$$\exists x (\forall y \sim \; p(x,y))$$

Thus we arrive at the equivalence

$$\sim \forall x \exists y \, p(x,y) \;=\; \exists x \forall y \sim \; p(x,y).$$

2.2.8 Expressions of predicate logic

We have defined predicates and quantifiers in previous sections. Now we define propositions and expressions of predicate logic.

Definition (propositions)
The propositions of predicate logic are intended to represent relationships between objects.

- The truth constants
  $true$ and $false$ are propositions.
- The propositional letters
  $P, Q, R, S, P_1, Q_1, R_1, S_1, P_2, ...$ are propositions.
- If $c_1, c_2, \ldots, c_n$ are constants, where $n \geq 1$, and p is an n -ary predicate, then
  $p(c_1, c_2, \ldots, c_n)$
  is a proposition.

Definition (expressions)
The *expressions* of predicate logic are built from its propositions according to the following rules:

- Every proposition is an expression.
- If $v_1, v_2, \ldots, v_n$ are variables or constants, where $n \geq 1$, and p is an n -ary predicate, then
  $p(v_1, v_2, \ldots, v_n)$
  is an expression.
- If $\mathcal{F}$ is an expression, then so is its negation
  $\sim \mathcal{F}$
- If $\mathcal{F}$ and $\mathcal{G}$ are expressions, then so are the following connections of them:
  $\mathcal{F} \wedge \mathcal{G}$
  $\mathcal{F} \vee \mathcal{G}$
  $\mathcal{F} \rightarrow \mathcal{G}$
  $\mathcal{F} \equiv \mathcal{G}$
- If v is any variable and $\mathcal{F}$ is an expression, then
  $\forall v \mathcal{F}$
  $\exists v \mathcal{F}$
  $\exists ! v \mathcal{F}$
  are expressions.

Example
Suppose p is a ternary predicate and q is a binary predicate. Then:

$p(x, a, b)$ and $q(x, y)$ are expressions;

$\exists y q(x, y)$ is an expression;

$p(x, a, b) \wedge \exists y q(x, y)$ is an expression; thus

$\forall x(p(x, a, b) \wedge \exists y q(x, y))$
is an expression.

Before we can consider the meaning of expressions, we must introduce the notions of bound and free variables in an expression. We begin with an example.
In the expression

$$\forall x(p(x, y) \wedge \exists y q(x, y))$$

there are two occurrences of x in the scope of the universal quantifier $\forall x$ both of which are bound as defined in the previous subsection. On the other hand, although there are also two occurrences of y, the occurrence in $p(x, y)$ is not within any quantifier of the form $\forall y$ or $\exists y$ so it is a free occurrence of y. The occurrence of y in $q(x, y)$ is a bound occurrence because it is within the scope of the quantifier $\exists y$.

Note that a variable can be within the scope of more that one quantifier. For example, in the expression

$$\forall x(p(x, y) \wedge (\exists x \forall y q(x, y)))$$

the final occurrence of x in $q(x, y)$ is in the scope of both the inner quantifier $\exists x$ and the outer quantifier $\forall x$. In this case, we regard x as being bound by the inner quantifier $\exists x$.

We note that renaming any bound variable does not change the value of the statement, so that, for example the expression

$$\forall x(p(x, y) \wedge (\exists x \forall y q(x, y)))$$

is equivalent to the expression in which the outermost occurrence of x is renamed z

$$\forall z(p(z,y) \wedge (\exists x \forall y q(x,y))).$$

Definition (bound and free occurrences)
Let v be a variable and $\mathcal{F}$ be an expression of predicate logic. The occurrence of v is bound in $\mathcal{F}$ if it is within the scope of a quantifier in $\mathcal{F}$; it is *bound by* the innermost quantifier that contains the occurrence of v within its scope. The occurrence of v is free in $\mathcal{F}$ if it is not within the scope of any quantifier in $\mathcal{F}$.

Example
In the expression

$$\mathcal{F}: \ \forall x(p(x,y) \wedge \exists y q(x,y,z))$$

both occurrences of x are bound by the quantifier $\forall x$. The first occurrence of y is free, while the second occurrence of y is bound by the quantifier $\exists y$. The occurrence of z is free in $\mathcal{F}$.

Definition (bound and free variables)
The variable v is bound in the expression $\mathcal{F}$ if there is at least one bound occurrence of v in $\mathcal{F}$, and free in $\mathcal{F}$ if there is at least one free occurrence of v in $\mathcal{F}$.

Definition (closed expression)

An expression is *closed* if it has no free occurrence of any variable.

Example
The expression

$$\forall x(p(x,y) \wedge \exists y q(y,z))$$

is not closed because the occurrence of z is free. On the other hand, the expression

$$\forall x \exists y p(x,y)$$

is closed.

2.2.9 Validity of expressions

In predicate logic, we define validity only for closed expressions, i.e. expressions with no free variables. Any closed expression, as has been shown in the case of binary predicates, becomes a proposition. Therefore, the definition of validity is the same as it is for propositional logic expressions.

Definition (valid)
A closed expression $\mathcal{F}$ is valid if it is true under every value of its components.

Establishing validity
Here, we do not introduce formal methods for proving the validity of closed expressions of predicate logic. However, we can use informal methods and common sense to convince ourselves that expressions are valid.

Examples

- Suppose we want to show the validity of the following expression

$$\sim (\forall x p(x)) \equiv \exists x(\sim p(x)).$$

According to the definition of $\equiv$, it suffices to show that

$$\sim \forall x p(x) \text{ and } \exists x(\sim p(x))$$

have the same truth value, i.e. the first expression is true precisely when the second is true. Suppose, we have that

$$\sim \forall x p(x) = \mathrm{T}$$

then we have by the definition of negation

$$\forall x p(x) = \mathrm{F}$$

The last statement is true (by the definition of the universal quantifier) precisely when there exists a domain element d such that

$$p(d) = \mathrm{F}$$

Hence,

$$\sim p(d) = \mathrm{T}$$

Hence,

$$\exists x \sim p(x) = \mathrm{T}$$

This result does not contain the element d taken in the assumption made above. Therefore,

$$\sim \forall x p(x) \;\Rightarrow\; \exists x \sim p(x)$$

Similarly, we can arrive at

$$\exists x \sim p(x) \;\Rightarrow \sim \forall x p(x)$$

From the last two statements we conclude that the expression

$$\sim \forall x p(x) \;\equiv\; \exists x \sim p(x)$$

is valid.

- Suppose we want to show the validity of the expression

$$\mathcal{F} : (\exists x(p(x) \wedge q(x)) \rightarrow (\exists x p(x)) \wedge (\exists x q(x))$$

It suffices (by the definition of the connective $\rightarrow$) to show that whenever the antecedent

$$\exists x(p(x) \wedge q(x))$$

is true, the consequent

$$\exists x p(x) \wedge \exists x q(x)$$

is also true.
Assume that

$$\exists x(p(x) \wedge q(x)) = \mathrm{T}$$

Then there exists a domain element, say d, such that $p(d) \wedge q(d)$ is true, i.e.

$$p(d) \wedge q(d) = \mathrm{T}$$

Hence (by the definition of $\wedge$)

$$p(d) = \mathrm{T} \text{ and } q(d) = \mathrm{T}$$

Hence,

$$\exists x p(x) = \mathrm{T} \text{ and } \exists x q(x) = \mathrm{T}$$

From these two statements we get (by the definition of $\wedge$)

$$\exists x\, p(x) \wedge \exists x q(x) = \mathrm{T}$$

as desired.

- Show the validity of the expression

$$\mathcal{F} : (\forall x \forall y\, p(x, y)) \rightarrow (\forall y \forall x\, p(x, y))$$

It suffices (by the definition of the connective $\rightarrow$) to show that, if the antecedent

$$\forall x \forall y\, p(x, y)$$

is true, then the consequent

$$\forall y \forall x\, p(x, y)$$

must also be true.
Assume that

$$\forall x \forall y\, p(x, y) = \mathrm{T}$$

Then, for any domain element d the following holds:

$$\forall y\, p(d, y) = \mathrm{T}$$

Similarly, for any domain element c the following holds

$$p(d, c) = \mathrm{T} \text{ for any } d \text{ and any } c$$

Therefore it follows that

$$\forall x\, p(x, c) = \mathrm{T} \text{ for any } c$$

Hence,

$$\forall y(\forall x\, p(x, y)) = \mathrm{T}$$

as desired.

3 Set theory basics

A fundamental concept in all branches of mathematics is that of the set, where a set can be thought of as a simple collection of objects. We have briefly used set theory notation in the section on Predicate logic (Section 2.2.2).

In this section, we present the basics of set theory.

3.1 Introduction

Definition (sets and elements)
A *set* is a collection of objects which can be distinguished from one another. The objects are called the *elements* or *members* of the set.

A set can contain any objects and in particular all the elements of a set need not be the same kind of object.

Definition (symbols)
The following symbols are traditionally used in set theory:

- The elements
  a, b, c, a_1, b_1, c_1, a_2, b_2, c_2, ...
- The sets
  $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{A}_1$, $\mathbf{B}_1$, $\mathbf{C}_1$, $\mathbf{A}_2$, ...

Definition (membership)
The elements of a set belong to the set or are members of the set. All other objects are not elements of the set and do not belong to the set.

We write the statements

- 'the element a belongs to the set $\mathbf{S}$',
- 'the element b does not belong to the set $\mathbf{S}$'

symbolically as

$$a \in \mathbf{S}$$

and

$$b \notin \mathbf{S}$$

respectively. The expression $a \in \mathbf{S}$ is also read as 'a is an element of the set $\mathbf{S}$', or 'a is contained in the set $\mathbf{S}$', or simply 'a is in (the set) $\mathbf{S}$'.

Similarly, the expression $a \notin \mathbf{S}$ is read as 'a is not an element of the set **S**', or 'a is not contained in the set **S**', or simply 'a is not in (the set) **S**'.

Definition (finite and infinite sets)
A *finite set* is one which includes only a finite number of elements, and an *infinite set* is one which includes an infinite number of elements.

Examples

- The set containing the integers from -1 to 1 inclusive is a finite set because its elements are the numbers -1, 0, 1.

Some well-known infinite sets are:

- **Z** – the set of all integers.
- **N** – the set of natural numbers, that is all non-negative integers.
- **Q** – the set of rational numbers. The rational numbers are all numbers that can be expressed as a fraction of the form $\frac{p}{q}$, where p is an integer and q is a natural number. Notice that this set includes all the integers since any integer z can be written as $\frac{z}{1}$. Other numbers in this set include all proper fractions (e.g. $\frac{1}{2}$, $\frac{37}{21}$), all terminating decimals (e.g. 0.5, 2.123) and all repeating decimals (e.g. 0.3333...).
- $\mathbf{Q}_1$ – the set of irrational numbers, that is those numbers that are not rational. This includes, for example, the numbers π, e, $\sqrt{2}$ and $\sqrt{3}$.
- **R** – the set of all real numbers, which consists of all rational and irrational numbers.

Defining sets

We often use the following ways to define a set:

1. **Enumeration**
   In this form, we simply write the elements of the set between braces { }. For example, **A**={2, 4, 6, 8} is the set containing the numbers 2, 4, 6 and 8.

   As mentioned above, a set is completely defined by the different elements it contains. Consequently, the order in which its elements appear, or the number of times that a particular element appears has no impact on the set. So, for example,

$\mathbf{A} = \{2, 4, 6, 8\} = \{8, 4, 6, 2\} = \{2, 4, 6, 8, 2\}$.

However, there is a difference between a set and the element in the set. For example, $\{moon\}$ denotes the set containing only the element *moon*. Consequently, $\{moon\} \neq moon$: one is a set containing a single object, the other is the object.

Furthermore, an element can be anything, including another set. For example, $\{\{1\}, \{2\}\}$ denotes the set whose elements are two sets, one containing only the number 1 and the other containing only the number 2.

2. **Using set comprehension**
Often it is difficult or impossible to actually list all the elements of a set explicitly. For example, the set of all natural numbers from 10 to 200 is simply too large to write in this way. In such cases we can define the set instead using set comprehension, which basically defines the set by defining the properties of its elements.

Definition (set comprehension)
Let $\mathbf{S}$ be a set and $p(x)$ be a predicate defined on the set $\mathbf{S}$. Then the expression

$$\{v \in \mathbf{S} \mid p(v)\}$$

denotes the set of all elements v of $\mathbf{S}$ for which $p(v)$ is true.

$\{v \in \mathbf{S} \mid p(v)\}$ is read as "the set of all v in $\mathbf{S}$ such that $p(v)$".

Examples

$\{n \in \mathbf{Z} \mid -2 \leq n < 3\}$	is the set of integers between -2 and 3, including -2; i.e. $\{-2, -1, 0, 1, 2\}$
$\{n \in \mathbf{N} \mid 10 \leq n \leq 200\}$	is the set of natural numbers from 10 to 200 inclusive.
$\{x \in \mathbf{R} \mid -10 < x < 10\}$	is the set of real numbers between -10 and 10. This is an open interval, which means that the endpoints -10 and 10 are not included.
$\{x \in \mathbf{Z} \mid x^2 = 4\}$	is the set of integers x with the property that x^2 equals 4; in other words, the set $\{-2, 2\}$

Definition (the universal set)
The *universal set* is the largest set within a given *universe of discourse*; i.e. the set of all the

elements which make up different sets being considered. Typically, the universal set is represented by **U**.

Example
If the universe being discussed includes sets made up of rational numbers, integers and natural numbers, then the universal set might be defined to be the set of all real numbers.

Definition (the empty set)
The *empty set* is the set which contains no elements and is generally written as $\{\}$ or $\emptyset$.

It may seem that defining a set with no elements is unnecessary. However, the empty set is as important to problems regarding sets as 0 is to mathematical problems.

Example
The set $\{x \in \mathbf{R} \mid x^2 = -1\}$ is empty because there are no real numbers whose square equals -1 (because $x^2 \geq 0$ holds for all real numbers), i.e.

$$\{x \in \mathbf{R} \mid x^2 = -1\} = \{\}$$

3.2 Relationships between sets

Let **A** and **B** be arbitrary sets.

Definition (subset)
A is a *subset* of **B**, written $\mathbf{A} \subseteq \mathbf{B}$, if and only if every element of **A** is also an element of **B**.

We can symbolically write this as $\mathbf{A} \subseteq \mathbf{B} \;=\; \forall a(a \in \mathbf{A} \;\rightarrow\; a \in \mathbf{B})$.

According to this definition, however, any set **A** is a subset of itself. Therefore we introduce the term proper subset.

Definition (proper subset)
A is a *proper subset* of **B**, written $\mathbf{A} \subset \mathbf{B}$, if and only if **A** is a subset of **B** but **A** is not equal to **B** (in other words, there exists at least one element in **B** which is not in **A**).

Definition (not a subset of)
A is *not a subset* of **B**, written $\mathbf{A} \not\subseteq \mathbf{B}$, if and only if there exists at least one element e which is in **A** but not in **B**, that is $\exists e(e \in \mathbf{A} \wedge e \notin \mathbf{B})$.

Definition (set equality)
Two sets **A** and **B** are *equal*, denoted **A**=**B**, if $\mathbf{A} \subseteq \mathbf{B}$ and $\mathbf{B} \subseteq \mathbf{A}$

Definition (superset and proper superset)
A superset is the opposite of a subset. If **A** is a subset of **B**, then **B** is a *superset* of **A**, written $\mathbf{B} \supseteq \mathbf{A}$. Likewise, if **A** is a proper subset of **B**, then **B** is a *proper superset* of **A**, written $\mathbf{B} \supset \mathbf{A}$.

The relationships of two sets, one of which is a subset of the second one, are shown in Figure 4 with a Venn diagram (see Section 2.2.3).

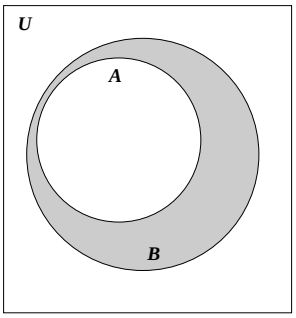

Figure 4: $\mathbf{A} \subset \mathbf{B}$ and $\mathbf{B} \supset \mathbf{A}$

The negations of all these properties of sets can be defined similarly, but we omit them here.

Example
Let $\mathbf{A} = \{n \in \mathbf{N} \mid n \leq 6\}$, $\mathbf{B} = \{1, 3, 7\}$, $\mathbf{C} = \{2, 4, 6\}$ and $\mathbf{D} = \{5, 2, 1, 3, 4, 6\}$. Then, for example, the following relationships hold:

- **B** is not a proper subset of **A**, i.e. $\mathbf{B} \not\subset \mathbf{A}$
- **C** is a subset of **A**, i.e. $\mathbf{C} \subseteq \mathbf{A}$
- **C** is a proper subset of **A**, i.e. $\mathbf{C} \subset \mathbf{A}$
- **A** and **D** are equal, i.e. **A**=**D**
- **A** is a proper superset of **C**, i.e. $\mathbf{A} \supset \mathbf{C}$

The following theorem and corollary hold for the empty set.

Theorem: The empty set is a subset of every set, that is $\emptyset \subseteq \mathbf{A}$ for any set $\mathbf{A}$.

Proof (by contradiction):
Suppose that $\emptyset \not\subseteq \mathbf{A}$. By our definition of 'not a subset of', this means

$$\exists x(x \in \emptyset \wedge x \notin \mathbf{A}).$$

However, $\emptyset$ has no elements, so $x \in \emptyset$ is false, and hence $x \in \emptyset \wedge x \notin \mathbf{A}$ is false. Thus, the statement $\exists x(x \in \emptyset \wedge x \notin \mathbf{A})$ is false, so the theorem is true.

Corollary: There is only one empty set, i.e. the empty set is unique.

Proof:
Let $\emptyset_1$ and $\emptyset_2$ be sets with no elements. Then, by the above theorem, $\emptyset_1 \subseteq \emptyset_2$ and also $\emptyset_2 \subseteq \emptyset_1$. So, by the definition of set equality $\emptyset_1 = \emptyset_2$.

3.3 Cardinality and power set

Definition (cardinality)
The *cardinality of a set* is simply the number of elements in the set. The cardinality of a set $\mathbf{S}$ is denoted by $|\mathbf{S}|$ or *card* $\mathbf{S}$.

By the definition of proper subset, it is clear that if $\mathbf{A} \subset \mathbf{B}$ then the set $\mathbf{A}$ must have fewer elements than $\mathbf{B}$ if $\mathbf{B}$ is finite, which yields the following statement:

$$\mathbf{A} \subset \mathbf{B} \Rightarrow |\mathbf{A}| < |\mathbf{B}|$$

Definition (power set)
The *power set* of a set $\mathbf{A}$ is the set of all subsets of $\mathbf{A}$. It is written as $P(\mathbf{A})$ or 2^A.

For any set $\mathbf{A}$, both $\{\}$ and $\mathbf{A}$ itself are elements of $P(\mathbf{A})$.

For any set $\mathbf{A}$ the following equality holds:

$$| P(\mathbf{A}) | = 2^{|\mathbf{A}|}$$

Example
If $\mathbf{A} = \{1, 2, 3\}$ then $P(\mathbf{A}) = \{\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$

3.4 Operations on sets

New sets can be formed from existing ones in a variety of ways. We describe some of these, as well as other relationships between sets which can be constructed from them, here.

Definition (union)
The *union* of two sets $\mathbf{A}$ and $\mathbf{B}$ is the set of all elements which are in either $\mathbf{A}$ or $\mathbf{B}$ or both. It is written as $\mathbf{A} \cup \mathbf{B}$.
Symbolically,

$$\mathbf{A} \cup \mathbf{B} = \{x \in \mathbf{U} \mid x \in \mathbf{A} \vee x \in \mathbf{B}\}$$

Note that for any set $\mathbf{A}$ the following hold:

$$\mathbf{A} \cup \{\} = \mathbf{A} \text{ and } \mathbf{A} \cup \mathbf{U} = \mathbf{U}.$$

Definition (intersection)
The *intersection* of two sets $\mathbf{A}$ and $\mathbf{B}$ is the set of all elements which are in both $\mathbf{A}$ and $\mathbf{B}$. It is written as $\mathbf{A} \cap \mathbf{B}$.
Symbolically,

$$\mathbf{A} \cap \mathbf{B} = \{x \in \mathbf{U} \mid x \in \mathbf{A} \wedge x \in \mathbf{B}\} = \{x \in \mathbf{A} \mid x \in \mathbf{B}\}$$

Note that for any set $\mathbf{A}$ the following hold:

$$\mathbf{A} \cap \{\} = \{\} \text{ and } \mathbf{A} \cap \mathbf{U} = \mathbf{A}.$$

Definition (disjoint)
If $\mathbf{A} \cap \mathbf{B} = \{\}$, then $\mathbf{A}$ and $\mathbf{B}$ are said to be *disjoint* sets. In other words, there is no element in $\mathbf{A}$ which is also in $\mathbf{B}$.

Definition (distributive union and distributive intersection)
The *distributive union* and *distributive intersection* of sets can be thought of as being generalisations of the union and intersection operators to an arbitrary number of sets. Distributive union

of sets A_1, A_2, ... , A_n, denoted by $\bigcup_{i=1}^{n} A_i$, is the set constructed by taking the union of all the elements in all the sets A_i $(1 \leq i \leq n)$. Thus, an object is an element of the set $\bigcup_{i=1}^{n} A_i$ if it is an element of some set A_i $(1 \leq i \leq n)$.

Distributive intersection of sets A_1, A_2, ... , A_n, denoted by $\bigcap_{i=1}^{n} A_i$, is the set constructed by taking the intersection of all the sets A_i $(1 \leq i \leq n)$. Thus, an object is an element of the set $\bigcap_{i=1}^{n} A_i$ if it is an element of all sets A_i $(1 \leq i \leq n)$.

Definition (complement)
The *complement* or *absolute complement* of a set $\mathbf{A}$, denoted by $\overline{\mathbf{A}}$ (or $\sim \mathbf{A}$), is the set of all elements that are not in $\mathbf{A}$.
Symbolically,

$$\overline{\mathbf{A}} = \{x \in \mathbf{U} \mid x \notin \mathbf{A}\}$$

Note that this definition of the complement of a set requires the existence of a Universal set.

Definition (difference)
The *difference* of two sets $\mathbf{A}$ and $\mathbf{B}$, denoted by $\mathbf{A} - \mathbf{B}$ (or $\mathbf{A} \backslash \mathbf{B}$), is the set of elements which are in $\mathbf{A}$ but not in $\mathbf{B}$. This is also known as the *complement of* $\mathbf{B}$ *relative to* $\mathbf{A}$.
Symbolically,

$$\mathbf{A} - \mathbf{B} = \{x \in \mathbf{U} \mid x \in \mathbf{A} \wedge x \notin \mathbf{B}\} = \{x \in \mathbf{A} \mid x \notin \mathbf{B}\}.$$

Note that $\mathbf{A} - \mathbf{B}$ is not equal to $\mathbf{B} - \mathbf{A}$ in general.
The absolute complement of a set $\mathbf{A}$ can now be defined as $\overline{\mathbf{A}} = \mathbf{U} - \mathbf{A}$.

Definition (boolean sum)
The *boolean sum* of two sets $\mathbf{A}$ and $\mathbf{B}$ is the set $\mathbf{A} + \mathbf{B}$ consisting of all elements which are in $\mathbf{A}$ or in $\mathbf{B}$ but not in both.
Symbolically,

$$\mathbf{A} + \mathbf{B} = \{x \in \mathbf{U} \mid (x \in \mathbf{A} \wedge x \notin \mathbf{B}) \vee (x \notin \mathbf{A} \wedge x \in \mathbf{B})\} = (\mathbf{A} - \mathbf{B}) \cup (\mathbf{B} - \mathbf{A})$$

Note that if the two sets $\mathbf{A}$ and $\mathbf{B}$ are disjoint then $\mathbf{A} + \mathbf{B} = \mathbf{A} \cup \mathbf{B}$

The union and intersection of two sets can be represented with Venn diagrams in the same way as the truth sets of $p(x) \vee q(x)$ and $p(x) \wedge q(x)$ were represented in Section 2.2.3. The difference and boolean sum of two sets are shown in Figure 5.

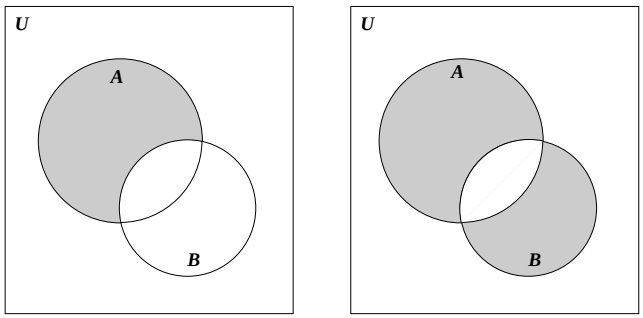

Figure 5: The difference and boolean sum of sets **A** and **B**

Definition (ordered pair)
An *ordered pair* is a pair of objects given in a fixed order. It is written as (a, b) where a is called the first component and b the second component of the pair.

Definition (product sets)
The *product set* of two sets **A** and **B** is the set of ordered pairs whose first component is taken from **A** and whose second component is taken from **B**:

$$\mathbf{A} \times \mathbf{B} = \{(x, y) \mid x \in \mathbf{A} \wedge y \in \mathbf{B}\}$$

It is also called the *Cartesian product*, or simply the *product* of **A** and **B**, and $\mathbf{A} \times \mathbf{B}$ is read "**A** cross **B**".

Examples

1. Let **U** be $\{n \in \mathbf{Z} \mid 0 \leq n \leq 11\}$,
   $\mathbf{A} = \{n \in \mathbf{U} \mid n < 7\}$, and $\mathbf{B} = \{n \in \mathbf{U} \mid 3 < n \leq 9\}$. Then
   - $\overline{\mathbf{A}} = \{n \in \mathbf{U} \mid n \geq 7\}$
   - $\overline{\mathbf{B}} = \{0, 1, 2, 3, 10, 11\}$
   - $\mathbf{A} \cup \mathbf{B} = \{n \in \mathbf{U} \mid n \leq 9\}$
   - $\mathbf{A} \cap \mathbf{B} = \{n \in \mathbf{U} \mid 3 < n < 7\}$
   - $\mathbf{A} - \mathbf{B} = \{n \in \mathbf{U} \mid n \leq 3\}$
   - $\mathbf{B} - \mathbf{A} = \{7, 8, 9\}$
   - $\mathbf{A} + \mathbf{B} = \{0, 1, 2, 3, 7, 8, 9\}$

2. Let $\mathbf{A} = \{a, b, c\}$ and $\mathbf{B} = \{1, 2, 3\}$ then
   $\mathbf{A} \times \mathbf{B} = \{(a,1), (a,2), (a,3), (b,1), (b,2), (b,3), (c,1), (c,2), (c,3)\}$

3.5 Laws of set algebra

We often need to prove theorems (properties) about sets.

Suppose, for example, we need to prove the equality

$$\overline{\mathbf{A} \cup \mathbf{B}} = \overline{\mathbf{A}} \cap \overline{\mathbf{B}}$$

This can be done as follows:

Let $x \in \overline{\mathbf{A} \cup \mathbf{B}}$. Then $x \notin (\mathbf{A} \cup \mathbf{B})$. Hence, $x \notin \mathbf{A}$ and $x \notin \mathbf{B}$. Hence, $x \in \overline{\mathbf{A}}$ and $x \in \overline{\mathbf{B}}$.
Hence, $x \in (\overline{\mathbf{A}} \cap \overline{\mathbf{B}})$.

Therefore, $\overline{\mathbf{A} \cup \mathbf{B}} \subseteq \overline{\mathbf{A}} \cap \overline{\mathbf{B}}$.

Now suppose, $x \in (\overline{\mathbf{A}} \cap \overline{\mathbf{B}})$. Then $x \in \overline{\mathbf{A}}$ and $x \in \overline{\mathbf{B}}$.
Hence, $x \notin \mathbf{A}$ and $x \notin \mathbf{B}$. Hence, $x \notin (\mathbf{A} \cup \mathbf{B})$. Hence, $x \in \overline{\mathbf{A} \cup \mathbf{B}}$.

Therefore, $\overline{\mathbf{A}} \cap \overline{\mathbf{B}} \subseteq \overline{\mathbf{A} \cup \mathbf{B}}$.

Thus, from

$$\overline{\mathbf{A} \cup \mathbf{B}} \subseteq \overline{\mathbf{A}} \cap \overline{\mathbf{B}}$$

and

$$\overline{\mathbf{A}} \cap \overline{\mathbf{B}} \subseteq \overline{\mathbf{A} \cup \mathbf{B}}$$

we conclude that

$$\overline{\mathbf{A}} \cap \overline{\mathbf{B}} = \overline{\mathbf{A} \cup \mathbf{B}}$$

This method of proving theorems is called an *element-wise proof*.

Set algebra is the branch of mathematics that uses only the laws and logic of algebra to prove theorems about sets. These laws simplify the study of sets and with them we can prove properties much more easily than using element-wise proofs.

Some algebraic laws of sets are listed below.

Laws
Let $\mathbf{U}$ be a universal set, and $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ be given sets. Then the following properties hold:

- Idempotent laws
  $\mathbf{A} \cap \mathbf{A} = \mathbf{A}$
  $\mathbf{A} \cup \mathbf{A} = \mathbf{A}$
- Associative laws
  $(\mathbf{A} \cap \mathbf{B}) \cap \mathbf{C} = \mathbf{A} \cap (\mathbf{B} \cap \mathbf{C})$
  $(\mathbf{A} \cup \mathbf{B}) \cup \mathbf{C} = \mathbf{A} \cup (\mathbf{B} \cup \mathbf{C})$
- Commutative laws
  $\mathbf{A} \cap \mathbf{B} = \mathbf{B} \cap \mathbf{A}$
  $\mathbf{A} \cup \mathbf{B} = \mathbf{B} \cup \mathbf{A}$
- Distributive laws
  $\mathbf{A} \cap (\mathbf{B} \cup \mathbf{C}) = (\mathbf{A} \cap \mathbf{B}) \cup (\mathbf{A} \cap \mathbf{C})$
  $\mathbf{A} \cup (\mathbf{B} \cap \mathbf{C}) = (\mathbf{A} \cup \mathbf{B}) \cap (\mathbf{A} \cup \mathbf{C})$
- Absorptive laws
  $\mathbf{A} \cap (\mathbf{A} \cup \mathbf{B}) = \mathbf{A}$
  $\mathbf{A} \cup (\mathbf{A} \cap \mathbf{B}) = \mathbf{A}$
- Identity laws
  $\mathbf{A} \cap \mathbf{U} = \mathbf{A}$
  $\mathbf{A} \cup \{\} = \mathbf{A}$
  $\mathbf{A} \cap \{\} = \{\}$
  $\mathbf{A} \cup \mathbf{U} = \mathbf{U}$
- Complement laws
  $\mathbf{A} \cap \overline{\mathbf{A}} = \{\}$
  $\mathbf{A} \cup \overline{\mathbf{A}} = \mathbf{U}$
  $\overline{\overline{\mathbf{A}}} = \mathbf{A}$
  $\overline{\mathbf{U}} = \{\}$
  $\overline{\{\}} = \mathbf{U}$

- DeMorgan's laws
  $\overline{\mathbf{A} \cup \mathbf{B}} = \overline{\mathbf{A}} \cap \overline{\mathbf{B}}$.
  $\overline{\mathbf{A} \cap \mathbf{B}} = \overline{\mathbf{A}} \cup \overline{\mathbf{B}}$.
- Alternative set difference representation
  $\mathbf{A} - \mathbf{B} = \mathbf{A} \cap \overline{\mathbf{B}}$
- Inclusion in union
  $\mathbf{A} \subseteq \mathbf{A} \cup \mathbf{B}$
  $\mathbf{B} \subseteq \mathbf{A} \cup \mathbf{B}$
- Inclusion in intersection
  $\mathbf{A} \cap \mathbf{B} \subseteq \mathbf{A}$
  $\mathbf{A} \cap \mathbf{B} \subseteq \mathbf{B}$
- Transitive properties of subsets
  If $\mathbf{A} \subseteq \mathbf{B}$ and $\mathbf{B} \subseteq \mathbf{C}$ then $\mathbf{A} \subseteq \mathbf{C}$

Examples (proofs)

- For any sets $\mathbf{A}$ and $\mathbf{B}$ we have

$$\mathbf{A} \cup (\overline{\mathbf{A}} \cap \mathbf{B}) = \mathbf{A} \cup \mathbf{B}$$

Proof:

$$\begin{array}{rcll} \mathbf{A} \cup (\overline{\mathbf{A}} \cap \mathbf{B}) & = & (\mathbf{A} \cup \overline{\mathbf{A}}) \cap (\mathbf{A} \cup \mathbf{B}) & \textit{distributive law} \\ & = & \mathbf{U} \cap (\mathbf{A} \cup \mathbf{B}) & \textit{complement law} \\ & = & \mathbf{A} \cup \mathbf{B} & \textit{identity law} \end{array}$$

- Show that

$$\mathbf{A} \cap (\mathbf{B} \cup \mathbf{A}) = \mathbf{A}$$

Proof:

$$\begin{array}{rcll} \mathbf{A} \cap (\mathbf{B} \cup \mathbf{A}) & = & (\mathbf{A} \cup \{\}) \cap (\mathbf{B} \cup \mathbf{A}) & \textit{identity law} \\ & = & (\mathbf{A} \cup \{\}) \cap (\mathbf{A} \cup \mathbf{B}) & \textit{commutative law} \\ & = & \mathbf{A} \cup (\{\} \cap \mathbf{B}) & \textit{distributive law} \\ & = & \mathbf{A} \cup \{\} & \textit{identity law} \\ & = & \mathbf{A} & \textit{identity law} \end{array}$$

- Prove that

$$\mathbf{A} = (\mathbf{A} \cup \mathbf{B}) \cap ((\mathbf{A} \cap \mathbf{B}) \cup \overline{\mathbf{B}})$$

Proof:

$$
\begin{array}{lll}
 & (\mathbf{A} \cup \mathbf{B}) \cap ((\mathbf{A} \cap \mathbf{B}) \cup \overline{\mathbf{B}}) & \\
= & (\mathbf{A} \cup \mathbf{B}) \cap ((\mathbf{A} \cup \overline{\mathbf{B}}) \cap (\mathbf{B} \cup \overline{\mathbf{B}})) & \textit{distributive law} \\
= & (\mathbf{A} \cup \mathbf{B}) \cap ((\mathbf{A} \cup \overline{\mathbf{B}}) \cap \mathbf{U}) & \textit{complement law} \\
= & (\mathbf{A} \cup \mathbf{B}) \cap (\mathbf{A} \cup \overline{\mathbf{B}}) & \textit{identity law} \\
= & \mathbf{A} \cup (\mathbf{B} \cap \overline{\mathbf{B}}) & \textit{distributive law} \\
= & \mathbf{A} \cup \{\} & \textit{complement law} \\
= & \mathbf{A} & \textit{identity law}
\end{array}
$$

4 Mappings

A way to introduce a relationship between the elements of two sets is mappings which are widely used in mathematics and computer science. Therefore, we present the notation of mappings in this section.

4.1 Introduction

Definition (mapping)
Let $\mathbf{A}$ and $\mathbf{B}$ be sets. A *mapping* from set $\mathbf{A}$ to set $\mathbf{B}$ is a relationship between all elements of $\mathbf{A}$ and some or all elements of $\mathbf{B}$ in which each element of $\mathbf{A}$ is related to a unique element of $\mathbf{B}$.

More precisely, a mapping can be thought of as a triple: the *source* is a set of objects; the *range* is another set of objects; and the *relation* is a subset $\mathbf{S}$ of the Cartesian product of the source with the range, such that for each element s of the source there is exactly one element r of the range such that the pair (s, r) lies in $\mathbf{S}$. Sometimes, such a mapping is referred to as a *total mapping*.

Definition (image)
For a mapping from $\mathbf{A}$ to $\mathbf{B}$, if s is an element of $\mathbf{A}$ (i.e. $s \in \mathbf{A}$) and r is the element of $\mathbf{B}$ that is related to s in the mapping, then r is called the *image* of s under the mapping, and we say that the mapping *maps* s to r and denote this by $s \mapsto r$.

Let m be a mapping from $\mathbf{A}$ to $\mathbf{B}$ and let $a \in \mathbf{A}$. Then the image of the element a under m is denoted by $m(a)$ and according to the definition of mapping this must be some element of $\mathbf{B}$, i.e. the following holds: $\forall a \, (a \in \mathbf{A} \rightarrow m(a) \in \mathbf{B})$

Definition (domain and range)

For a mapping from **A** to **B** the set **A** is called the *domain* and the set **B** the *range* of the mapping.

We denote the domain of a mapping m by $dom(m)$ and its range by $rng(m)$.

Definition (finite and infinite mappings)
A mapping is *finite* if its domain is a finite set and *infinite* if its domain is an infinite set.

Defining mappings
We often use the following two ways to define a mapping:

1. **Enumeration**
   We write a mapping by listing its associations between brackets [ ] explicitly. This is similar to the enumeration form of a set (see Section 3.1) because a mapping is a set of unordered associations.

   Examples

   - $m_1 = [A \mapsto 65, B \mapsto 66, C \mapsto 67, a \mapsto 97, b \mapsto 98, c \mapsto 99]$

   is the mapping which maps A to 65, B to 66, C to 67, a to 97, etc.

   Its domain and range are the sets $dom(m_1) = \{A, B, C, a, b, c\}$ and $rng(m_1) = \{65, 66, 67, 97, 98, 99\}$ respectively.

   The image of A is 65, and the image of a is 97.

   Note that the order in which the associations are listed does not matter. So, the mapping written above can equivalently be written as:

   $m_1 = [A \mapsto 65, a \mapsto 97, B \mapsto 66, b \mapsto 98, C \mapsto 67, c \mapsto 99]$

   - For the mapping $m_2 = [2 \mapsto true, 3 \mapsto false, 4 \mapsto true, 5 \mapsto false]$

   its domain and range are $dom(m_2) = \{2, 3, 4, 5\}$ and $rng(m_2) = \{true, false\}$ respectively.

2. **Map comprehension**
   We can define a mapping using a notation similar to the one for set comprehension based on the fact that a mapping is a set of associations. The notation

$$[f(s) \mapsto g(s) \mid s \in \mathbf{S} \wedge p(s)]$$

represents the set of associations of the form $f(s) \mapsto g(s)$, where f and g are some functions of s, for all s belonging to the set $\mathbf{S}$ which satisfy the given predicate p.

Examples

$[n \mapsto -n \mid n \in \mathbf{Z} \wedge (-1 \leq n \leq 1)] = [-1 \mapsto 1, 0 \mapsto 0, 1 \mapsto -1]$

$[n \mapsto 2 \cdot n - 1 \mid n \in \mathbf{N} \wedge (n \leq 4)] = [0 \mapsto -1, 1 \mapsto 1, 2 \mapsto 3, 3 \mapsto 5, 4 \mapsto 7]$

It is impossible to define an infinite mapping by enumeration, while it is possible to create an infinite mapping using map comprehension. For example, the mapping

$[n \mapsto 2 \cdot n - 1 \mid n \in \mathbf{N} \wedge (n > 4)]$

is an infinite mapping because its domain is the set

$\{n \in \mathbf{N} \mid n > 4\}$

which is infinite.

Definition (the empty mapping)
The *empty* mapping is the mapping whose domain is the empty set and which thus contains no associations. It is represented by [ ].

For example, the mapping

$[n \mapsto 2 \cdot n - 1 \mid n \in \mathbf{N} \wedge (n < 0)]$

is the empty mapping because its domain is the set

$\{n \in \mathbf{N} \mid n < 0\}$

which is empty.

Properties of mappings

Definition (injective mapping)
A mapping is *injective* if for each element r of the range of the mapping there is at most one element of the domain whose image under the mapping is r.

Definition (surjective mapping)
A mapping is *surjective* if for each element r of the range of the mapping there is at least one element of the domain which maps to r under the mapping.

Definition (bijection)
A mapping is a *bijection* if it is both injective and surjective. Then for each element r of the range of the mapping there is exactly one element of the domain that maps to r. We also say that the mapping is *bijective* or a *one-to-one correspondence*.

Definition (inverse mapping)
For a bijection m its *inverse mapping* is the mapping that maps each element e of $rng(m)$ to the unique element of $dom(m)$ that maps to e. We denote this by m^{-1}. Then

$$m^{-1} = [m(a) \mapsto a \mid a \in dom(m)]$$

Example
Let $\mathbf{B} = \{a, b, c, d\}$,
$\mathbf{A} = \{1, 2, 3\}$,
$\mathbf{A}_1 = \{1, 2, 3, 4, 5\}$ and
$\mathbf{A}_2 = \{1, 2, 3, 4\}$, and let
$m_1 = [1 \mapsto a, 2 \mapsto b, 3 \mapsto c]$ be a mapping from $\mathbf{A}$ to $\mathbf{B}$,
$m_2 = [1 \mapsto a, 2 \mapsto b, 3 \mapsto a]$ also be a mapping from $\mathbf{A}$ to $\mathbf{B}$,
$m_3 = [1 \mapsto a, 2 \mapsto a, 3 \mapsto b, 4 \mapsto c, 5 \mapsto d]$ be a mapping from $\mathbf{A}_1$ to $\mathbf{B}$, and
$m_4 = [1 \mapsto d, 2 \mapsto c, 3 \mapsto b, 4 \mapsto a]$ be a mapping from $\mathbf{A}_2$ to $\mathbf{B}$.

Then the mapping m_1 is injective but not surjective and hence not bijective; the mapping m_2 is neither injective nor surjective, so also not bijective; the mapping m_3 is surjective but not injective, so not bijective; and the mapping m_4 is both injective and surjective and hence also bijective.

The inverse mapping of m_4 is $m_4^{-1} = [d \mapsto 1, c \mapsto 2, b \mapsto 3, a \mapsto 4]$.

Definition (permutation)
We define the notation of permutations as an example of a mapping, which we then use in our examples below. A *permutation* of the positive integers $1, 2, \ldots, n$ is a bijection from $\mathbf{N}_n$ to itself where $\mathbf{N}_n = \{i \in N \mid 1 \leq i \leq n\}$.

A more visual way of representing a permutation is to use a two row table (one row per copy of $\mathbf{N}_n$). The first row is the domain values and the second is the range values of the bijection. The values in both rows appear in their natural order and straight lines connect each value i to its image. Two such permutations are shown in Figure 6.

A *cycle* in a permutation is a sequence of associations in the permutation in which the domain value of each association, except the first, is equal to the image of the previous association and

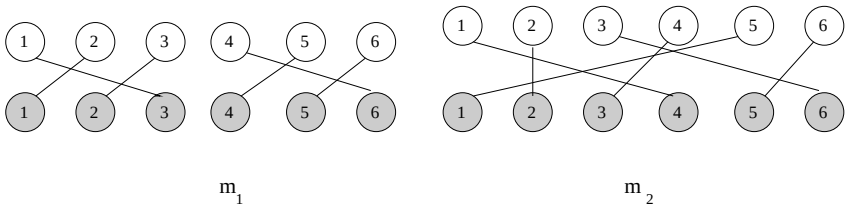

Figure 6: Permutations

the image of the last association is equal to the domain value of the first association. If there is an element which is mapped to itself then it is considered as one cycle. A cycle can be represented by listing only the domain values of the associations in it. We write this list in braces. For example, the cycles of the mapping m_1 shown in Figure 6 are (1,3,2) and (4,6,5), and the cycles of m_2 are (1,4,3,6,5) and (2).

So, another way to represent a permutation is using a *product of cycles* in which we write all cycles of the permutation without any delimiters. For example, the permutations represented in Figure 6 can be written as

$m_1 = (1,3,2)(4,6,5)$ and $m_2 = (1,4,3,6,5)(2)$.

4.2 Relationships between mappings

Definition (mapping equality)
Two mappings m_1 and m_2 are equal if and only if their domains are equal, their ranges are equal, and for each element s of the domain $m_1(s)$ is equal to $m_2(s)$. We write this simply as $m_1 = m_2$ and this is defined symbolically as
$(dom(m_1) = dom(m_2)) \wedge (rng(m_1) = rng(m_2)) \wedge \forall s(s \in dom(m_1) \rightarrow (m_1(s) = m_2(s)))$

4.3 Operations on mappings

Definition (product)
Let m_1 and m_2 be two mappings such that the range of m_1 is a subset of the domain of m_2, i.e. $rng(m_1) \subseteq dom(m_2)$. Then the *product* of the two mappings is the mapping which maps each element s of the domain $dom(m_1)$ to the value which is the image of $m_1(s)$ under the mapping m_2, i.e. to $m_2(m_1(s))$. It is denoted by $m_1 * m_2$.

The product of two mappings is sometimes called the *composition* of the mappings.

Example
The product of the two mappings m_1 and m_2 given above in Figure 6 is $m_1 * m_2 = (1,6)(2,4,5,3)$ and is illustrated in Figure 7.

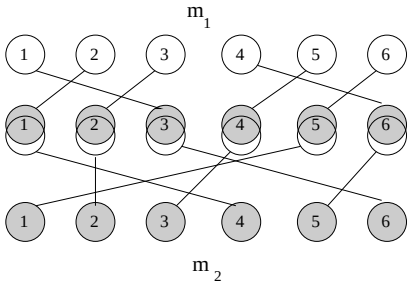

Figure 7: The product of two mappings

Definition (power)
Let m be a mapping, the range of which is a subset of its domain, and let $n \in \{i \in \mathbf{N} \mid i \geq 1\}$. Then $m\hat{}n$ is the n-fold composition of the mapping m.

Example
$m_1\hat{}3 = (1)(2)(3)(4)(5)(6)$. This is illustrated in Figure 8.

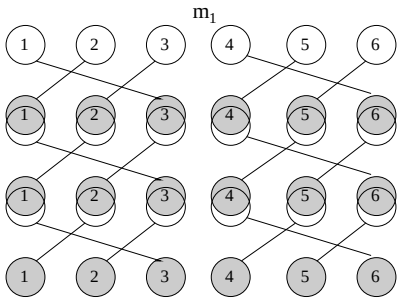

Figure 8: 3-fold composition of a mapping

Definition (override)
The *override* operator overrides one mapping with another where priority is given to the associations in the second operand when the domain values match. It is written $m_1 \dagger m_2$ where m_1 and m_2 are mappings.

Example
Let $m_1 = [1 \mapsto 3, 2 \mapsto 1, 3 \mapsto 2, 4 \mapsto 6, 5 \mapsto 4, 6 \mapsto 5]$. Then $m_1 \dagger [2 \mapsto 3, 3 \mapsto 2, 4 \mapsto 7] = [1 \mapsto 3, 2 \mapsto 3, 3 \mapsto 2, 4 \mapsto 7, 5 \mapsto 4, 6 \mapsto 5]$. This is illustrated in Figure 9.

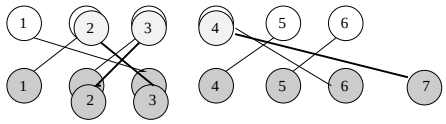

Figure 9: Overriding of mappings

Definition (union)
The *union* operator combines two mappings whose domains are disjoint, simply building the set union of the associations in the two mappings. It is written $m_1 \cup m_2$ where m_1 and m_2 are mappings.

Example
$[1 \mapsto 3, 2 \mapsto 1, 3 \mapsto 2] \cup [4 \mapsto 6, 5 \mapsto 4, 6 \mapsto 5] = [1 \mapsto 3, 2 \mapsto 1, 3 \mapsto 2, 4 \mapsto 6, 5 \mapsto 4, 6 \mapsto 5]$

Definition (domain subtraction)
The *domain subtraction* operator removes all the associations from a mapping whose domain values are in a given set. It is written $m \setminus s$ where m is a mapping and s is a set.

Example
Let $m = [1 \mapsto 3, 2 \mapsto 1, 3 \mapsto 2, 4 \mapsto 6, 5 \mapsto 4, 6 \mapsto 5]$. Then $m \setminus \{2, 3, 4, 7\} = [1 \mapsto 3, 5 \mapsto 4, 6 \mapsto 5]$ which is illustrated in Figure 10.

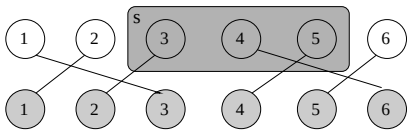

Figure 10: Restricting a mapping by a given set

Definition (domain restriction)
The *domain restriction* operator removes all the associations from a mapping whose domain values are not in a given set. It is written m / s where m is a mapping and s is a set.

Example
Let $m = [1 \mapsto 3, 2 \mapsto 1, 3 \mapsto 2, 4 \mapsto 6, 5 \mapsto 4, 6 \mapsto 5]$. Then $m / \{2, 3, 4, 7\} = [2 \mapsto 1, 3 \mapsto 2, 4 \mapsto 6]$. This is illustrated in Figure 11.

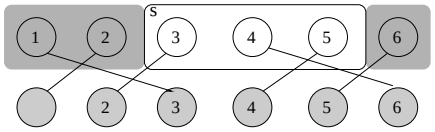

Figure 11: Restricting a mapping to a given set

Definition (range subtraction)
The *range subtraction* operator removes all the associations from a mapping whose range values are in a given set. It is written $m \ominus s$ where m is a mapping and s is a set.

Example
Let $m = [1 \mapsto 3, 2 \mapsto 1, 3 \mapsto 2, 4 \mapsto 6, 5 \mapsto 4, 6 \mapsto 5]$. Then $m \ominus \{2, 3, 4, 7\} = [2 \mapsto 1, 4 \mapsto 6, 6 \mapsto 5]$.

Definition (range restriction)
The *range restriction* operator removes all the associations from a mapping whose range values are not in a given set. It is written $m \oslash s$ where m is a mapping and s is a set.

Example
Let $m = [1 \mapsto 3, 2 \mapsto 1, 3 \mapsto 2, 4 \mapsto 6, 5 \mapsto 4, 6 \mapsto 5]$. Then $m \oslash \{2, 3, 4, 7\} = [1 \mapsto 3, 3 \mapsto 2, 5 \mapsto 4]$.

5 Functions

5.1 Introduction

Let $\mathbf{X}$ and $\mathbf{Y}$ be sets.

Definition (partial function)
A *partial function* f from $\mathbf{X}$ to $\mathbf{Y}$ is a subset of $\mathbf{X} \times \mathbf{Y}$ (the Cartesian product of $\mathbf{X}$ and $\mathbf{Y}$) which satisfies the condition that for each x in $\mathbf{X}$ there is at most one y in $\mathbf{Y}$ such that (x, y) is in f.

A given function f is undefined for a value $x \in \mathbf{X}$ if there is no $y \in \mathbf{Y}$ such that (x, y) is in f.

Definition (total function)
A partial function is called a *total function* (or simply *function*) if for each value x in $\mathbf{X}$ there is exactly one y in $\mathbf{Y}$ such that (x, y) is in f.

We present in this section the basic theory of functions.

A function can also be defined as follows:

Definition (function)
A function f from set $\mathbf{X}$ to set $\mathbf{Y}$ is a relationship between the elements of the sets $\mathbf{X}$ and $\mathbf{Y}$ in which each element of $\mathbf{X}$ is related to a unique element of $\mathbf{Y}$. It is denoted by $f : \mathbf{X} \to \mathbf{Y}$.

Note that according to this definition of functions any mapping as defined in Section 4.1 is a function.

Notation (function application)
We use the notation

$$f(x)$$

which is read "f *of* x", to represent the element y of $\mathbf{Y}$ which is related to x by the function f. The variable x is called the argument of the function f.

Definition (image and preimage)
Given an element a in $\mathbf{X}$, there is a unique element b in $\mathbf{Y}$ that is related to a and which is denoted by $f(a)$. We call this b the *image of* a under f, and a is called the preimage of b.

Definition (domain, co-domain and range)
For a function f from $\mathbf{X}$ to $\mathbf{Y}$ the set $\mathbf{X}$ is called the *domain* of f and $\mathbf{Y}$ is called the *co-domain* of f. The *range* of the function is the set of all images of elements in $\mathbf{X}$.

Definition (two variable function)
Let $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ be sets.

A function f from $\mathbf{X} \times \mathbf{Y}$ (the Cartesian product of the sets $\mathbf{X}$ and $\mathbf{Y}$) is a relationship between the elements of $\mathbf{X} \times \mathbf{Y}$ and the set $\mathbf{Z}$ where each element of $\mathbf{X} \times \mathbf{Y}$ is related to a unique element of $\mathbf{Z}$. It is denoted by $f : \mathbf{X} \times \mathbf{Y} \to \mathbf{Z}$.

We use the notation

$f(x, y)$

which is read "*f of x and y*" or "*f of x, y*", to represent the application of f to the pair (x, y). The variables x and y are called the arguments of the function f.

Definition (multi-variable function)
Let $n \in \mathbf{N}$ $(n \geq 2)$ and let $\mathbf{X}_1$, $\mathbf{X}_2$, ..., $\mathbf{X}_n$ and $\mathbf{Y}$ be sets.

A function f from $\mathbf{X}_1 \times \mathbf{X}_2 \times \cdots \times \mathbf{X}_n$ (the Cartesian product of the sets) is a relationship between the elements of $\mathbf{X}_1 \times \mathbf{X}_2 \times \cdots \times \mathbf{X}_n$ and the set $\mathbf{Y}$ where each element of $\mathbf{X}_1 \times \mathbf{X}_2 \times \cdots \times \mathbf{X}_n$ is related to a unique element of $\mathbf{Y}$. It is denoted by $f : \mathbf{X}_1 \times \mathbf{X}_2 \times \cdots \times \mathbf{X}_n \to \mathbf{Y}$.

The application of f to the arguments $x_1, x_2, \ldots, x_n$ is written as

$f(x_1, x_2, \ldots, x_n)$.

5.2 Defining functions

Functions can be defined in several different ways:

1. **Arrow diagram**
   If $\mathbf{X}$ and $\mathbf{Y}$ are finite sets, an *arrow diagram* depicts a function f from $\mathbf{X}$ to $\mathbf{Y}$ by drawing an arrow from each element in $\mathbf{X}$ to its image in $\mathbf{Y}$. In this case, two properties must hold for the diagram according to the definition of functions:

   - every element x of $\mathbf{X}$ must have an arrow coming out of it.
   - No element of $\mathbf{X}$ can have two or more arrows coming out of it.

   An example of an arrow diagram is shown in Figure 12.

2. **Table**
   If $\mathbf{X}$ and $\mathbf{Y}$ are finite sets, a two row *table* can be used to represent a function f from $\mathbf{X}$ to $\mathbf{Y}$. The first row contains all the different values of x in $\mathbf{X}$ and the second shows the image of x under f in each case.

   Example
   A function is defined by the following table:

x	A	B	C	a	b	c
y	65	66	67	97	98	99

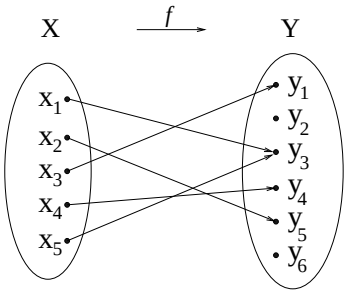

Figure 12: A function represented by an arrow diagram

3. **Analytical form**
When the domain of a function is an infinite set it is impossible to represent the function using the two methods described above. A more general way of defining a function is to give a formula defining the result of the function $f(x)$ in terms of the argument x. This formula is called the *analytical form* of the function.

Examples

- Suppose a function f is defined by the following table:

x	-2	-1	0	1	2
y	0	1	2	3	4

  Then its analytic form might be $f(x) = x + 2$ defined on the set $\{-2, -1, 0, 1, 2\}$.

- **Identity function on a set**
The *identity function* on a set $\mathbf{X}$

$$i_x : \mathbf{X} \to \mathbf{X}$$

  is defined by $i_x(x) = x$.

- **Absolute value function**
is the function $f : \mathbf{R} \to \mathbf{R}$ (denoted by $|x|$) such that

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

- **Polynomial function of degree n**
is a function $f : \mathbf{R} \to \mathbf{R}$ such that

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

  where $a_i \ \ (0 \leq i \leq n)$ are real constants.

- **Exponential function**
is a function $f : \mathbf{R} \to \mathbf{R}^+$ such that

$$f(x) = a^x$$

where $\mathbf{R}^+ = \{r \in \mathbf{R} \mid r > 0\}$ and a is a real constant such that $a > 0$ and $a \neq 1$.

4. **Function machines**
   Functions can be thought of as machines or computer programs. Suppose f is a function from $\mathbf{X}$ to $\mathbf{Y}$ and an input x of $\mathbf{X}$ is given. Then f can be imagined as a machine (or program) that processes x in a certain way to produce the output $f(x)$.

   Example
   The *Hamming distance function* was invented by the computer scientist Richard W. Hamming. It gives a measure of the "distance" between two strings of 0's and 1's that have the same length.

   Let $\mathbf{S} = \{0, 1\}$ and $n \in \mathbf{N}$ and let $\mathbf{S}^n$ be the set of all string of 0's and 1's of length n. Then the Hamming distance function $h : \mathbf{S}^n \times \mathbf{S}^n \to \mathbf{N}$ returns, for a pair in $\mathbf{S}^n \times \mathbf{S}^n$, the number of positions at which the two strings of the pair have different values, i.e.

   $$h(s, t) = \text{the number of positions at which } s \text{ and } t \text{ have different values.}$$

   Suppose $n = 6$. Then $\mathbf{S}^6$ is the set of strings comprising any combination of six 0's and 1's, and we can apply the distance function to two strings in the set $\mathbf{S}^6$. For example,

   $$h(110001, 110010) = 2$$

   because 110001 and 110010 differ only in the last two positions.

   A computer program (or procedure) which evaluates the distance between two such strings is a function.

5.3 Classification of functions

Definition (one-to-one function)

The function $f : \mathbf{X} \to \mathbf{Y}$ is called a *one-to-one function* (or *injection*) if, and only if, any two distinct elements x_1 and x_2 of $\mathbf{X}$ have distinct images under f. This can be stated symbolically as:

$$f(x_1) = f(x_2) \Rightarrow x_1 = x_2$$

or equivalently as

$$x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)$$

Conversely, a function $f : \mathbf{X} \to \mathbf{Y}$ is not a one-to-one function if there exist distinct elements x_1 and x_2 in $\mathbf{X}$ such that x_1 and x_2 have the same image under f, i.e. if $f(x_1) = f(x_2)$ with $x_1 \neq x_2$.

In terms of arrow diagrams, a one-to-one function takes distinct points of the domain to distinct points of the co-domain. A function is not a one-to-one function if at least two points of the domain are taken to the same point of the co-domain. An example of this is shown in Figure 13.

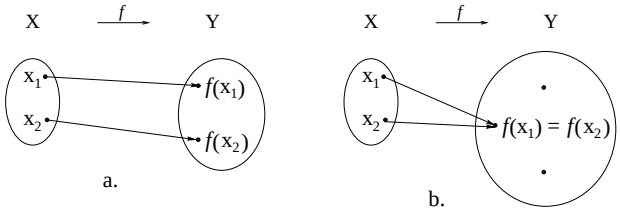

Figure 13: a. A one-to-one function. b. A function that is not one-to-one

One-to-one functions on infinite sets

To prove a function is one-to-one the method of direct proof is generally used. Consider the following examples.

Example: Let $f : \mathbf{R} \to \mathbf{R}$ be a function defined by the rule

$$f(x) = 6x - 2$$

Prove that f is one-to-one on $\mathbf{R}$.

Proof: Suppose x_1 and x_2 are real numbers such that $f(x_1) = f(x_2)$. (We need to show $x_1 = x_2$). Then

$$6x_1 - 2 = 6x_2 - 2$$

Adding 2 to both sides gives

$$6x_1 = 6x_2$$

Dividing by 6 on both sides we then arrive at

$$x_1 = x_2$$

as desired.

Example: Let $f : \mathbf{R} \to \mathbf{R}$ be a function defined by the rule

$$f(x) = 2x^2 + 1$$

Prove that f is not *one-to-one* on $\mathbf{R}$.

Proof: Suppose x_1 and x_2 are real numbers such that $f(x_1) = f(x_2)$. (We need to show x_1 and x_2 may be different). Then

$2x_1^2 + 1 = 2x_2^2 + 1$

Adding -1 to both sides gives

$2x_1^2 = 2x_2^2$

Dividing by 2 on both sides gives

$x_1^2 = x_2^2$

Hence, $x_1^2 - x_2^2 = 0$ or $(x_1 + x_2) \cdot (x_1 - x_2) = 0$

The product is equal to 0 if either the first or the second multiplier is equal to 0, i.e.

$$\left[\begin{array}{l} x_1 + x_2 = 0 \\ x_1 - x_2 = 0 \end{array}\right.$$

Hence, we arrive at

$$\left[\begin{array}{l} x_1 = -x_2 \\ x_1 = x_2 \end{array}\right.$$

From the first equation we conclude that if $x_2 \neq 0$ then $x_1 \neq x_2$ (for example, if $x_2 = 3$ then $x_1 = -3$) .

Hence, $f(x_1) = f(x_2)$ but $x_1 \neq x_2$, and therefore $f(x)$ is not one-to-one.

Definition (onto functions)
The function $f : \mathbf{X} \to \mathbf{Y}$ is said to be *onto* (or *surjective*) if, and only if, each element in $\mathbf{Y}$ is the image of some element of $\mathbf{X}$ under the function f, i.e.

f is onto if, and only if, $\forall y(y \in \mathbf{Y} \to \exists x(x \in \mathbf{X} \wedge f(x) = y))$ is true.

Conversely, a function $f : \mathbf{X} \to \mathbf{Y}$ is not onto if there is some y in $\mathbf{Y}$ which is not the image of any x in $\mathbf{X}$, i.e. if $\exists y(y \in \mathbf{Y} \ \wedge \forall x(x \in \mathbf{X} \wedge f(x) \neq y))$ is true.

Example
Let $f: \mathbf{R} \to \mathbf{R}$ be a function defined by the rule

$$f(x) = 6x - 2$$

Prove that f is onto on $\mathbf{R}$.

Proof: Let $y \in \mathbf{R}$.
We need to show that there exists $x \in \mathbf{R}$ such that $f(x) = y$. If such a real number exists, then $6x - 2 = y$ or $x = (y + 2)/6$. x is a real number since sums and quotients (except for division by 0) of real numbers are real. It follows that f is onto.

Example
Let $f: \mathbf{R} \to \mathbf{R}$ be a function defined by the rule

$$f(x) = 2x^2 + 1$$

Prove that f is not onto on $\mathbf{R}$.

Proof: We need to show that there is some value $y \in \mathbf{R}$ for which there does not exist $x \in \mathbf{R}$ such that $f(x) = y$.
Consider $y = -5$. Then if $f(x) = y$ for some x we require

$$\begin{aligned} 2x^2 + 1 &= -5 \text{ or} \\ 2x^2 + 1 &= -6 \text{ or} \\ x^2 &= -3 \end{aligned}$$

which has no solution for $x \in \mathbf{R}$

Hence, the function f is not onto.

A function $f: \mathbf{X} \to \mathbf{Y}$ can be both *one-to-one* and *onto*. In this case for any element x in $\mathbf{X}$ there is a unique corresponding element $y = f(x)$ in $\mathbf{Y}$: for any element y in $\mathbf{Y}$ there is some element x in $\mathbf{X}$ such that $f(x) = y$ because f is *onto*, and there is only one such element x because f is *one-to-one*.

In this case, the function f sets up a relationship between the elements of $\mathbf{X}$ and $\mathbf{Y}$ in which each element of $\mathbf{X}$ relates to exactly one element of $\mathbf{Y}$ and each element of $\mathbf{Y}$ relates to exactly

one element of **X**. Thus, we give the following definition.

Definition (one-to-one correspondence)
A function is called a *one-to-one correspondence* or *bijection* if it is both *one-to-one* and *onto*.

An example is shown in Figure 14.

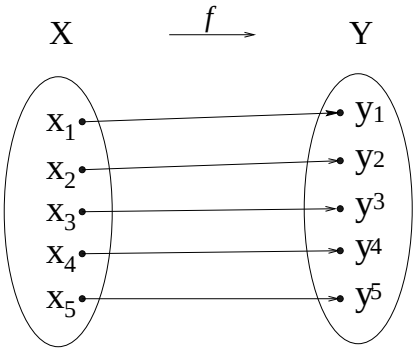

Figure 14: A function which is a one-to-one correspondence

If f is a one-to-one correspondence from a set **X** to a set **Y**, then there is a function from **Y** to **X** under which each element y in **Y** is related to the unique element x in **X** which is the preimage of y under the function f. This function is called the inverse function of f.

Definition (inverse function)
Suppose $f: \mathbf{X} \to \mathbf{Y}$ is a one-to-one correspondence. Then the function $f^{-1}: \mathbf{Y} \to \mathbf{X}$ defined as

$$f^{-1}(y) = x \quad \text{if, and only if} \quad y = f(x)$$

is the *inverse function* of f.

The diagram in Figure 15 shows that an inverse function "sends" each element back to where it came from.

Finding the inverse function of a function given by a formula
Suppose $f(x) = y$ is a one-to-one correspondence and is given by a formula. Then it has an inverse function. To find this function we usually rewrite the formula so as to express x in terms

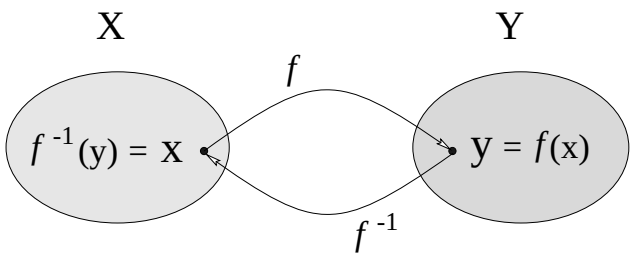

Figure 15: A function f and its inverse function f^{-1}

of y.

Example
Let $f:\ \mathbf{R} \to \mathbf{R}$ be a function defined by the rule

$$f(x) = 6x - 2$$

It has already been shown above that f is *one-to-one* and *onto*. Hence f is a one-to-one correspondence and has an inverse function f^{-1}. Find the function f^{-1}.

Solution: By the definition of f^{-1}

$$f^{-1}(y) = x \text{ whenever } f(x) = y$$

But $f(x) = 6x - 2$. So, we have $6x - 2 = y$.
Solving this equation for x, we arrive at

$$x = \frac{y+2}{6}$$

Hence $f^{-1}(y) = \frac{y+2}{6}$.

5.4 Composition of functions

Let $g:\ \mathbf{X} \to \mathbf{Y}_1$ and $f:\ \mathbf{Y} \to \mathbf{Z}$ be functions such that $\mathbf{Y}_1 \subseteq \mathbf{Y}$.

Definition (composition of functions)
The *composition* of functions f and g, which is denoted by $f \circ g$, is a function from $\mathbf{X}$ to $\mathbf{Z}$ such that

$$(f \circ g)(x) = f(g(x)) \text{ for all } x \text{ in } \mathbf{X}.$$

The diagram in Figure 16 demonstrates the composition of two functions.

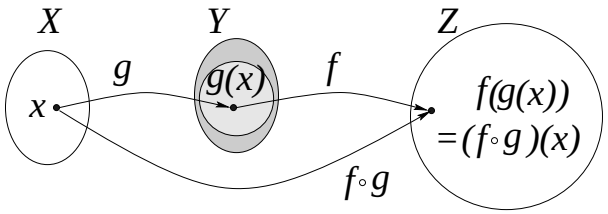

Figure 16: Composition of two functions

Example
Let $g : \mathbf{R} \to \mathbf{R}$ be defined by $g(x) = x^2 + 2$, and let $f : \mathbf{R} \to \mathbf{R}$ be defined by $f(x) = 3x + 4$. Then

$$(f \circ g)(x) = f(g(x)) = 3(x^2 + 2) + 4 = 3x^2 + 6 + 4 = 3x^2 + 10$$

$$(g \circ f)(x) = g(f(x)) = (3x + 4)^2 + 2 = 9x^2 + 24x + 16 + 2 = 9x^2 + 24x + 18.$$

Function composition satisfies two important properties:

1. **Composition with the identity function**
   Let g be the identity function. Then $f \circ g = f$ holds for every function f.

2. **Composing a function with its inverse**
   Let f be a one-to-one correspondence. Then

   $$f \circ f^{-1} \text{ and } f^{-1} \circ f$$

   are identity functions.

6 Relations

6.1 Introduction

The term 'relation' is used to describe a relationship between one object and another. In the case we are discussing, the 'one object and another' happen to be the elements of sets. For example, given elements a and b in $\mathbf{R}$, $a < b$, $a = b$, and $a^2 + 1 > b$ are all examples of the many legitimate relationships between these two elements.

Definition (n-ary relation)
An *n-ary relation* R between sets A_1, A_2, ... , A_n is a subset of $A_1 \times A_2 \times \cdots \times A_n$:

$$\mathrm{R} = \{(x_1, x_2, \ldots, x_n) \in A_1 \times A_2 \times \cdots \times A_n \mid p(x_1, x_2, \ldots, x_n)\}$$

where $n \in \mathbf{N}$ $(n \geq 2)$, and p is a predicate representing the properties of the elements of the relation R.

n is called the *arity* of the relation R.

Definition (binary relation)
A relation R of arity 2 is called a *binary relation* from A_1 to A_2. A_1 is called the *domain* and A_2 the *co-domain* of the relation. If (x, y) is in R, we denote this by $x\mathrm{R}y$.

Definition (relation on a set)
If $S = A_1 = A_2 = \cdots = A_n$ for a relation R, then R is called a *relation on the set S*.

Examples

- Let $\mathbf{A} = \{1, 2, 3\}$ and $\mathbf{B} = \{4, 5, 6\}$, and let R be a binary relation defined on $\mathbf{A} \times \mathbf{B}$ as follows:

  $$\{(x, y) \mid y/x \in \mathbf{Z}\}$$

  Then $R = \{(1, 4), (1, 5), (1, 6), (2, 4), (2, 6), (3, 6)\}$.

  Like functions, binary relations can be represented by arrow diagrams. To create an arrow diagram for the above example, we create regions for $\mathbf{A}$ and $\mathbf{B}$ and list the elements of the

sets as points. Then we draw an arrow from each x in **A** to each y in **B** where $(x, y) \in$ R (see Figure 17).

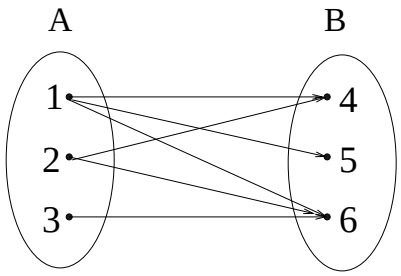

Figure 17: A relation represented by an arrow diagram

- Let $\mathbf{S} = \{-2, -1, 0, 1, 2\}$, and let R be a relation on **S** defined as follows:

$$\{(x, y) \mid x = -y \vee x/y = -2\}$$

Then $R = \{(-2, 2), (-1, 1), (0, 0), (1, -1), (2, -2), (2, -1), (-2, 1)\}$.

Because the relation is defined on a single set, an arrow diagram for R becomes a *directed graph*. Instead of creating two regions and mapping from one region to the other we list the elements of **S** as points in a single region and draw an arrow between points which are related to each other under R (see Figure 18). In such a directed graph the elements of **S** are called *vertices* and the elements of R (i.e. the ordered pairs that are in R) are *edges*; the vertices joined by an edge are the *endpoints* of the edge; an edge with just one endpoint is called a *loop*. There are 5 vertices and 7 edges, one of which is a loop, in Figure 18.

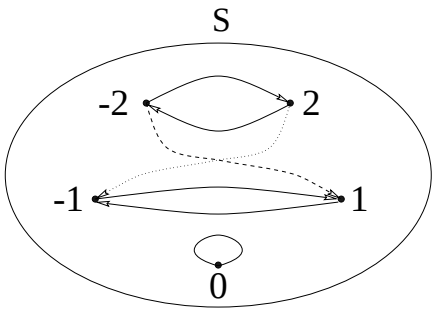

Figure 18: A binary relation on a set

- Let $p(x, y, z)$ be a predicate on **R** such that 'x, y and z are the coordinates of the vertices of a cube, the ends of one diagonal of which are the points (0, 0, 0) and (1, 1, 1)', and let a ternary relation be defined as

$$\mathrm{R} = \{(x, y, z) \in \mathbf{R} \times \mathbf{R} \times \mathbf{R} \mid p(x, y, z)\}$$

Then R is equal to

$$\{(0,0,0), (1,0,0), (1,1,0), (0,1,0), (0,0,1), (1,0,1), (1,1,1), (0,1,1)\}$$

6.2 Functions and relations

In what follows in this section, we focus on binary relations on a set **S**.

According to the definition, a binary relation is a subset of a Cartesian product, and Cartesian products are defined as sets of ordered pairs. Therefore, binary relations can be defined using only set theory. We have already defined functions in Section 5.1. Now it is possible to define a function in terms of binary relations as follows:

Definition (functions in terms of binary relations)
A function f from set **X** to set **Y** is a relation with the following two properties:

1. for every element x in **X**, there exists an element y in **Y** such that $(x, y) \in f$;
2. for all elements x in **X** and y and z in **Y**, if $(x, y) \in f$ and $(x, z) \in f$, then $y = z$; i.e. each element x in **X** is related to a unique element y in **Y**.

It is important to remember that the elements in a relation are ordered pairs, so that x and y must be used in the correct order. $f(x) = y$ if and only if y is the second element of the pair in f of which x is the first element.

A relation can also be defined as a function as follows.

Let **Bool** be the set $\{true, false\}$.

Definition (binary relations in terms of functions)
A binary relation on a set **S** is a function $r : \mathbf{S} \times \mathbf{S} \to \mathbf{Bool}$ such that $r(x, y)$ is *true* for every $(x, y) \in \mathbf{S} \times \mathbf{S}$ for which x and y are related and *false* otherwise.

Example
Let $\mathbf{S} = \{-2, -1, 0, 1, 2\}$, and let R be a relation on **S** as follows:

$$\{(x, y) \mid x = -y \vee x/y = -2\}$$

This relation can be written as the following function:

$$r(x, y) = \text{`}x = -y \vee x/y = -2\text{'}$$

which is true for each pair in the set
$\{(-2, 2), (-1, 1), (0, 0), (1, -1), (2, -2), (2, -1), (-2, 1)\}$
but false for all other pairs in $\mathbf{S} \times \mathbf{S}$.

6.3 Classification of binary relations

Let r be a binary relation on a set $\mathbf{S}$, i.e.

$$r : \mathbf{S} \times \mathbf{S} \to \mathbf{Bool}$$

Definition (reflexive relation)
r is *reflexive* if and only if $r(x, x)$ is true for every element x in $\mathbf{S}$, i.e. if $\forall x(x \in \mathbf{S} \to r(x, x))$ then r is reflexive.

Definition (irreflexive relation)
r is *irreflexive* if and only if $r(x, x)$ is false for every element x in $\mathbf{S}$.

Note that r can be neither reflexive nor irreflexive.

Definition (symmetric relation)
r is *symmetric* if and only if $r(x, y) = r(y, x)$ for every x and y in $\mathbf{S}$.

Definition (antisymmetric relation)
r is *antisymmetric* if and only if $r(x, y)$ and $r(y, x)$ are both true only if $x = y$ for every x and y in $\mathbf{S}$.

This means that, for an antisymmetric relation on $\mathbf{S}$, only one of $r(x, y)$ and $r(y, x)$ can be true for all distinct elements x and y in $\mathbf{S}$.

Note that r can be both symmetric and antisymmetric, and r can be neither symmetric nor antisymmetric.

Definition (transitive relation)
r is *transitive* if and only if whenever $r(x, y)$ and $r(y, z)$ are both true, $r(x, z)$ is also true for every x, y and z in $\mathbf{S}$.

In the directed graph representation these properties correspond to the following properties of the graph:

reflexive:	Every vertex has a loop.
irreflexive:	No vertex has a loop.
symmetric:	If there is an edge from one vertex to another, then there is an edge in the opposite direction.
antisymmetric:	There is at most one edge between distinct vertices.
transitive:	If there is an indirect path linking two vertices via one or more intermediate vertices, then there is also an edge linking the two vertices directly.

Example
Consider the following relations on $\mathbf{Z}$.

$\mathrm{R}_1 = \mathbf{Z} \times \mathbf{Z}$
$\mathrm{R}_2 = \{(x, y) \in \mathbf{Z} \times \mathbf{Z} \mid x = y\}$
$\mathrm{R}_3 = \{(x, y) \in \mathbf{Z} \times \mathbf{Z} \mid x < y\}$
$\mathrm{R}_4 = \{(x, y) \in \mathbf{Z} \times \mathbf{Z} \mid x \leq y\}$
$\mathrm{R}_5 = \{(x, y) \in \mathbf{Z} \times \mathbf{Z} \mid$ 'x is divisible by y'$\}$

The table below shows the properties of each of these relations:

	R_1	R_2	R_3	R_4	R_5
reflexive	√	√		√	√
irreflexive			√		
symmetric	√	√			
antisymmetric		√			
transitive	√	√	√	√	√

Definition (equivalence relation)
A binary relation on $\mathbf{S}$ is called an *equivalence relation* on $\mathbf{S}$ if and only if it is reflexive, symmetric and transitive.

Example
Let n be a positive integer and let R be the relation defined by

$$\mathrm{R}= \{(i,j) \in \mathbf{Z} \times \mathbf{Z} \mid \exists k(k \in \mathbf{Z} \wedge k \cdot n = i - j)\}$$

- R is reflexive, since $0 \cdot n = i - i$ for every $i \in \mathbf{Z}$.
- R is symmetric, since if $k \cdot n = i - j$ then $(-k) \cdot n = j - i$, i.e. if $(i,j) \in \mathrm{R}$ then $(j,i) \in \mathrm{R}$.
- R is transitive.
  Proof: Suppose that $(i,j) \in \mathrm{R}$ and $(j,m) \in \mathrm{R}$. Then there are integers k_1 and k_2 such that $k_1 \cdot n = i - j$ and $k_2 \cdot n = j - m$. Thus, we have

  $$k_1 \cdot n + k_2 \cdot n = (i - j) + (j - m) \text{ or}$$
  $$(k_1 + k_2) \cdot n = (i - m) \text{ where } k_1 + k_2 \text{ is an integer.}$$

  Hence, $(i,m) \in \mathrm{R}$.

Therefore, R is an equivalence relation on $\mathbf{Z}$.

Definition (order relations)
A *partial order* on $\mathbf{S}$ is a reflexive, antisymmetric and transitive binary relation on $\mathbf{S}$.

Elements x and y in $\mathbf{S}$ are said to be *comparable* in a relation R if and only if either (x,y) or (y,x) is in R.

A partial order on $\mathbf{S}$ is called a *total order* (or *linear order*) on $\mathbf{S}$ if and only if for every x and y in $\mathbf{S}$, x and y are comparable.

Example
The relation "less than or equal to" $\leq$ on $\mathbf{N}$ is a partial order.

Proof:

- **Reflexive:** For every $i \in \mathbf{N}$, $i \leq i$ is true.
- **Antisymmetric:** Both $i \geq j$ and $j \geq i$ hold only if $i = j$.
- **Transitive:** If $i \leq j$ and $j \leq k$ hold, then $i \leq k$ holds.

Thus, $\leq$ is a partial order on $\mathbf{N}$. In fact any two numbers are comparable, therefore $\leq$ is a total order.

6.4 Operations on relations

Let R_1 and R_2 be binary relations from $\mathbf{X}$ to $\mathbf{Y}$.

1. Relations are sets of ordered pairs. Therefore, we can define some operations on relations which are similar to the operations on sets.
   - **Union**
     $R_1 \cup R_2 = \{(x, y) \in \mathbf{X} \times \mathbf{Y} \mid (x, y) \in R_1 \vee (x, y) \in R_2\}$
   - **Intersection**
     $R_1 \cap R_2 = \{(x, y) \in \mathbf{X} \times \mathbf{Y} \mid (x, y) \in R_1 \wedge (x, y) \in R_2\}$
   - **Difference**
     $R_1 - R_2 = \{(x, y) \in \mathbf{X} \times \mathbf{Y} \mid (x, y) \in R_1 \wedge (x, y) \notin R_2\}$
   - **Complement**
     $\overline{R_1} = \mathbf{X} \times \mathbf{Y} - R_1\}$
2. **Inverse** (or **Converse**)
   The *inverse* R^{-1} of a given relation R from $\mathbf{X}$ to $\mathbf{Y}$ is defined by:
   $R^{-1} = \{(x, y) \in \mathbf{X} \times \mathbf{Y} \mid (y, x) \in R\}$
3. **Composition**
   Let R_1 be a binary relation from $\mathbf{X}$ to $\mathbf{Y}$ and R_2 be a binary relation from $\mathbf{Y}$ to $\mathbf{Z}$.
   The *composition* $R_1 \circ R_2$ of the two relations R_1 and R_2 is the relation defined as follows:
   $R_1 \circ R_2 = \{(x, z) \in \mathbf{X} \times \mathbf{Z} \mid \exists y((x, y) \in R_1 \wedge (y, z) \in R_2)\}$

   Example
   Let $R_1 = \{(a, a), (a, b), (b, d)\}$ and $R_2 = \{(a, d), (b, c), (b, d), (c, b)\}$ be relations on the set $\{a, b, c, d\}$.

   Then $R_1 \circ R_2 = \{(a, d), (a, c)\}$.

6.5 Transitive and reflexive closures

Suppose we are given a set $\mathbf{S} = \{a, b, c, d, e\}$, and a relation R defined on the set $\mathbf{S}$ as follows:

$$R = \{(a, e), (e, d), (d, c), (e, c), (c, b)\}$$

Consider the directed graph of R (see Figure 19). The given graph is not transitive. What associations would have to be added to make this directed graph transitive?

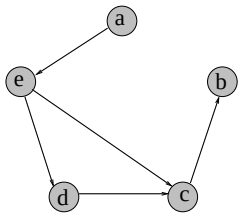

Figure 19: A given intransitive relation

Beginning from the vertex a and working around the graph, we would need to add the edges (a, d), (a, c), (e, b) and (d, b) to complete the triangles in this graph, the result of which is shown in Figure 20 (a). What we have done is add associations to the relation R to form a new relation, which we denote by R^+. Thus,

$$\mathrm{R}^+ = \{(a, e), (e, d), (a, d), (d, c), (e, c), (a, c), (c, b), (e, b), (d, b)\}$$

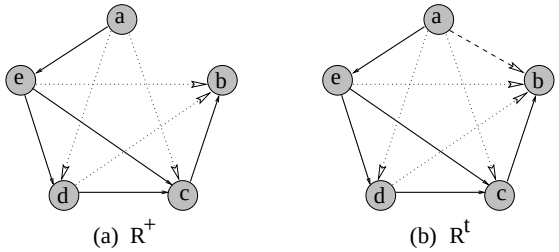

Figure 20: The transitive closure of the relation

Note that $\mathrm{R} \subseteq \mathrm{R}^+$, that is all pairs in R are also contained in R^+. However, R^+ is still not transitive. Adding new associations has created new subsets of elements which are not transitive. For example, $a\mathrm{R}d$ and $d\mathrm{R}b$, but $\sim (a\mathrm{R}b)$. Therefore, more edges must be added to create a transitive graph, as shown in Figure 20 (b). The new relation which has been created, which is denoted by R^t, is transitive.

The process of creating a transitive relation R^t from an intransitive relation R is accomplished by systematically adding new associations to R. More specifically, we wish to add the least number of associations possible in order to obtain a transitive relation. The relation created in this way is called the *transitive closure* of the relation R.

The transitive closure of a relation R satisfies the following properties:

1. R^t is transitive.

2. R is a subset of R^t, i.e. $R \subseteq R^t$.
3. If S is any other transitive relation such that $R \subseteq S$, then $R^t \subseteq S$, i.e. the transitive closure R^t is the minimal transitive relation containing the relation R.

Definition (reflexive and transitive closure)
Let R be a relation defined on a set **S**. Then, the *reflexive and transitive closure* of the relation R is the relation R^* defined on **S**, as follows:

$$R^* = \bigcup_{n \geq 0} R^n$$

where R^n are binary relations on **S** such that

$$sR^0t \text{ iff } s = t$$
$$sR^nt \text{ iff } \exists u(sR^{n-1}u \wedge uRt) \text{ if } n > 0$$

The following properties hold for these relations:

1. $R^1 = R$
2. sR^*t iff $\exists n \geq 0, \exists s_0, \ldots, \exists s_n \in \mathbf{S}$ with $s_0 = s$, $s_n = t$, and $s_i R s_{i+1}$ for all $i < n$.
3. sR^*t is reflexive and transitive.
4. If T is any reflexive and transitive relation on **S** such that $R \subseteq T$, then $R^* \subseteq T$. That is R^* is the smallest reflexive and transitive relation on **S** that contains R.

7 Recursion

7.1 Introduction

What is Recursion?
Recursion (in mathematics and in programming) is a technique for defining a problem in terms of one or more smaller versions of the same problem. The solution to the problem is built out of the results from the smaller versions.

Example
Let $n \in \mathbf{N}$. Then the following functions are *recursive* functions since each of them uses itself in its definition (or to compute its own value).

1. The factorial of n
$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n > 0 \end{cases}$$

2. The exponential function
$$a^n = \begin{cases} 1 & \text{if } n = 0 \\ a \cdot a^{n-1} & \text{if } n > 0 \end{cases}$$ where a is any real number ($a \neq 0$).

3. The *Fibonacci sequence*
$$F(n) = \begin{cases} 1 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n \geq 2 \end{cases}$$

In mathematics such functions are called *recurrences* and in programming a routine that calls itself is termed recursive.

Thus, recursion is the concept of well-defined self-reference. Self-referential definitions can be dangerous if we are not careful to avoid circularity. "A rose is a rose" is not a recurrence. Therefore, the definition of recursion should include the word *well-defined*.

Problems that lend themselves to recursive solutions have the following characteristics:

1. The problem can be redefined in terms of one or more subproblems, identical in nature to the original problem but in some sense smaller in size.
2. One or more base cases of the problem have direct or known solutions.
3. By applying this redefinition process to ever smaller subproblems, eventually the problem is reduced entirely to the base cases.
4. The base case solutions can be used in some way to build the solution to the whole problem.

A base case is an instance of the problem whose solution requires no further recursive definition (or call). It is a special case whose solution you know. Every recursive definition requires at least one base case in order to be valid. A base case has two purposes:

- It acts as a terminating condition. For example, without an explicitly defined base case a recursive routine would call itself indefinitely.
- It is the building block of the complete solution of the problem. For example, a recursive routine determines the final result from the base case it reaches.

Key concepts
When we attempt to construct a recursive solution of a problem we should keep in mind the following four questions:

1. How can we define the problem in terms of one or more smaller problems?
2. What instances of the problem can serve as the base cases?
3. As the problem size diminishes will we reach these base cases?
4. How are the solutions from the smaller problems used to build a correct solution to the larger problem?

It is not necessary or even desirable to ask ourselves the above questions in strict order. For example sometimes the solution to a problem is easier to envisage if we first ask ourselves what instances can serve as the base cases and then define the problem in terms of smaller problems of the same type which are closer to the base cases.

We demonstrate how to construct recursive functions in the next sections.

7.2 If expressions

The recursive definition of functions generally consists of an if statement with the following form:

if (this is a base case)
  then solve it directly
  else if (this is not a base case)
    redefine the problem using recursion
end if

Therefore, we begin by introducing if expressions which we then use in the following examples.

Definition (if expression)
An *if expression* is an expression of the form

if *pred* **then** *expr1* **else** *expr2* **end**

where the predicate *pred* is used to choose between the evaluation of two alternative expressions *expr1* and *expr2*.

If the predicate *pred* evaluates to *true* the expression *expr1* is evaluated, otherwise the expression *expr2* is evaluated. The two expressions *expr1* and *expr2* must be the same *type*, in the sense that these expressions take values from the same set. This type is also the type of the entire if expression.

Examples

- The expression

  $$\textbf{if } x \geq 0 \textbf{ then } x \textbf{ else } -x \textbf{ end}$$

  returns the absolute value of a real number x.

- The expression

  $$\textbf{if } n \geq m \textbf{ then } n \textbf{ else } m \textbf{ end}$$

  returns the greater of the two integers m and n.

If expressions can be nested as follows:

if $pred_1$
  then $expr_1$
  else
    if $pred_2$
      then $expr_2$
      else $expr_3$
    end
end

where all the expressions $expr_1$, $expr_2$ and $expr_3$ should be of the same type.

7.3 Explicit definition of functions

In Section 5.2 the absolute value function was defined as

$$f : \mathbf{R} \to \mathbf{R} \qquad \text{such that}$$
$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

The first expression is called the *signature* (or *header*), and the second is the expression defining the result of the function $f(x)$ in terms of the argument x. These two parts of the definition can be merged into one definition called an *explicit function definition*. For example, the absolute value function can be defined using an if expression as follows:

$$
\begin{array}{l}
f : \mathbf{R} \to \mathbf{R} \\
f(x) = \\
\quad \textbf{if} \;\; (x \geq 0) \\
\quad\quad \textbf{then} \;\; x \\
\quad\quad \textbf{else} \;\; -x \\
\quad \textbf{end} \\
\textbf{end} \;\; f
\end{array}
$$

where the variable x is called the *formal parameter* of f and takes any value from the domain **R**.

This style of defining functions is typical in programming languages and it is probably familiar to the reader. Therefore, we use it in our examples below.

7.4 Factorial function

Problem
The problem is to compute the factorial of a natural number n recursively.

Design
The familiar iterative definition of the factorial of n (or $n!$) is

$$
\begin{array}{ll}
n! = n \cdot (n-1) \cdot (n-2) \cdots 1 & \text{for} \;\; n > 0 \\
0! = 1
\end{array}
$$

We have four questions to answer:

1. How can we define the problem in terms of smaller problems of the same type?

   We need to define $n!$ in terms of the factorial of a smaller number:

$n! = n \cdot (n-1)!$ *for* $n > 0$

2. What instance of the problem can serve as the base case? The smallest number for which we can calculate the factorial is 0. Thus

   $n = 0$ *is the natural choice for our base case.*
   $0! = 1$ *by definition.*

3. As the problem size diminishes will we reach this base cases?

   Since each application of the function reduces the parameter n *by 1, and* n *is non-negative, we will always reach the base case* $n = 0$ *eventually.*

   Now, we define the function *factorial* as follows:

```
factorial : N → N
factorial(n) =
        if  (n = 0)
           then  1
           else  n * factorial(n − 1)
        end
end  factorial
```

4. How is the solution from the smaller problem used to build a correct solution to the larger problem?

   The result returned from the call to $factorial(n-1)$ *is multiplied by* n *to obtain* $factorial(n)$.

As you can see, the recursive function directly mimics the above recursive definition. This similarity between definition and implementation is a principal attraction of recursion.

7.5 Tracing a recursive function

Doing a trace by hand of multiple calls to a recursive function (for one or two simple examples) can be helpful in understanding how a recursion works. We note that it is less useful when trying to develop a recursive function (or algorithm).

In Table 1 a trace of the events in the evaluation of $factorial(3)$ is shown. We write all the events for a particular call to the function $factorial(n)$ in the same column. Here, "$\rightarrow$" denotes entry to and "$\leftarrow$" denotes exit from the function call named at the head of the column. (Recall

that we use T instead of *true* and F instead of *false*.)

$$
\begin{array}{llll}
factorial(3) & & & \\
\rightarrow 3 = 0 \ \ \mathrm{F} & & & \\
\qquad result\ is & 3 * factorial(2) & & \\
 & \rightarrow 2 = 0 \ \ \mathrm{F} & & \\
 & \qquad result\ is & 2 * factorial(1) & \\
 & & \rightarrow 1 = 0 \ \ \mathrm{F} & \\
 & & \qquad result\ is & 1 * factorial(0) \\
 & & & \rightarrow 0 = 0 \ \ \mathrm{T} \\
 & & & \qquad result\ is\ 1 \\
 & & & \leftarrow \ return(1) \\
 & & result\ is\ 1 * 1 & \\
 & & \leftarrow \ return(1) & \\
 & result\ is\ 2 * 1 & & \\
 & \leftarrow \ return(2) & & \\
result\ is\ 3 * 2 & & & \\
\leftarrow \ return(6) & & &
\end{array}
$$

Table 1: Tracing the function $factorial$

The number 6 is returned to the calling environment. Note that the multiplication operation is performed after the call to $factorial(n-1)$ returns a value.

7.6 The greatest common divisor

The greatest common divisor (gcd) of two integers is the largest integer that divides them both. We recall that $m\ mod\ n$ is the remainder of m divided by n; for example, $5\ mod\ 2 = 1$, $3\ mod\ 5 = 3$, $4\ mod\ 2 = 0$, $2\ mod\ 2 = 0$, etc.

Problem
The problem is to calculate the gcd of two non-negative integers m and n recursively.

Design
Euclid's algorithm for finding $gcd(m, n)$ can be defined recursively as follows:

$$gcd(m,n) = \begin{cases} m & \text{if } n = 0 \\ gcd(n, m\ mod\ n) & \text{if } n > 0 \end{cases}$$

Is this definition sufficient to be implemented as a recursive function? We consider the four questions again.

1. Clearly $gcd(m, n)$ has been defined in terms of a problem of the same type, but is $gcd(n, m \ mod \ n)$ smaller in size?

   The value of $(m \ mod \ n)$ lies in the range $0, 1, \ldots, n-1$. In other words, $(m \ mod \ n)$ is always less than n. Thus, if $m > n$ at the start, then $gcd(n, m \ mod \ n)$ is a smaller problem than $gcd(m, n)$.

   If $m \leq n$ at the start, then $(m \ mod \ n) = m$ and the first recursive step $gcd(n, m \ mod \ n)$ is equivalent to $gcd(n, m)$. This has the effect of exchanging the parameter values m and n. So after the first call we are back in the situation where the first parameter is greater than the second. Therefore, after the second recursive step $gcd(n, m \ mod \ n)$ would be smaller than the original problem.

2. If $n = 0$ then $gcd(m, n) = m$ by the definition of the function. So, our base case is when $n = 0$.

   $n = 0$ holds if and only if $(m \ mod \ n) = 0$ which means that n divides m.

3. As the parameters m and n decrease with every call (except maybe the first call as mentioned above) and $0 \leq (m \ mod \ n) < n$ we are sure to reach the base case $(m \ mod \ n) = 0$ eventually.

   Now, we define the function gcd as follows:

   $gcd : \ \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$
   $gcd(m, n) =$
   if $(n = 0)$
   then m
   else $gcd(n, m \ mod \ n)$
   end
   end gcd

   Note that this function returns 0 if both parameters are equal to 0. Thus this case should be explicitly considered in the calling environment.

4. How is the solution from the smaller problem used to build a correct solution to the larger problem?

   In this function the result from the smaller $gcd(n, m \ mod \ n)$ is the direct solution to the larger problem $gcd(m, n)$. All the function has to do is find the solution of the base case and return it unchanged until it reaches the original problem.

In Table 2 we show a trace of the events in the evaluation of $gcd(18, 48)$.

$gcd(18, 48)$
$\rightarrow 48 = 0$ F
result is $gcd(48, 18)$
$\rightarrow 18 = 0$ F
result is $gcd(18, 12)$
$\rightarrow 12 = 0$ F
result is $gcd(12, 6)$
$\rightarrow 6 = 0$ F
result is $gcd(6, 0)$
$\rightarrow 0 = 0$ T
result is 6
$\leftarrow$ *return*(6)
result is 6
$\leftarrow$ *return*(6)
result is 6
$\leftarrow$ *return*(6)
result is 6
$\leftarrow$ *return*(6)

Table 2: Tracing the function *gcd*

The number 6 is returned to the calling environment.

7.7 Intermediate recursion example

Let $\mathbf{N}_1$ be the set $\{i \in \mathbf{N} \mid n \geq 1\}$ and let $n \in \mathbf{N}_1$ be a given number.

Problem
A scientist wishes to make a *safe chain* of length n out of plutonium and lead pieces under the following conditions:

- No two plutonium pieces can be next to each other.
- Pieces of the same element are indistinguishable.
- There are at least n pieces of plutonium and n pieces of lead.
- Order is significant, i.e. the chains lead-plutonium and plutonium-lead, for example, are considered to be two different chains.

The problem is to compute recursively how many ways we can construct a safe chain of length n.

Design

1. How can we define the problem in terms of one or more smaller problems of the same type?

   There are two classes of safe chains: chains that end with a lead piece, and chains that end with a plutonium piece.

   Let $c(n)$ = the number of safe chains of length n,
   $l(n)$ = the number of safe chains of length n ending with a lead piece, and
   $p(n)$ = the number of safe chains of length n ending with a plutonium piece
   be functions on $\mathbf{N}_1$.
   Then, it is clear that $c(n) = l(n) + p(n)$.

   A safe chain of length n that ends with a lead piece is simply any safe chain of length $n-1$ with a lead piece tacked onto the end. Therefore,

   $$l(n) = c(n-1)$$

   A safe chain can end with a piece of plutonium if and only if the piece before it is a lead piece. That is, a safe chain of length n that ends with a plutonium piece is a safe chain of length $n-1$ that ends with lead. Therefore,

   $$p(n) = l(n-1) = c(n-2) \text{ (by substitution from above)}$$

   So, we arrive at

   $$c(n) = l(n) + p(n) = c(n-1) + c(n-2)$$

   This recursive relation introduces a new point: there may be cases where we solve a problem by solving more than one smaller problem of the same type.

2. What instances of the problem can serve as the base cases?
   In this case, we should be careful to define the base cases. If we simply say that $c(1)$ is the base case then what happens when $c(2)$ is called?
   $c(2) = c(1) + c(0)$, but $c(0)$ is undefined, which makes $c(2)$ undefined. Therefore, it is necessary to give $c(2)$ an explicit definition, i.e. to make it a second base case. Thus, the base cases are:

   $c(1) = 2$ because there are chains that consist of a single piece of lead or plutonium.

   $c(2) = 3$ because there are chains that consist of pieces: lead-plutonium, plutonium-lead, and lead-lead.

3. As the problem size diminishes will we reach the base cases? Since n is a positive integer and each call to the function reduces the parameter n by 1 or 2, we will always reach the base cases $n = 1$ and $n = 2$.

   The function can be defined as follows:

```
c : N1 → N
c(n) =
      if (n = 1)
        then 2
        else if (n = 2)
                then 3
                else c(n − 1) + c(n − 2)
             end
      end
 end c
```

4. How are the solutions from the smaller problems used to build a correct solution to the larger problem? The recursive step adds the results from the two smaller problems to obtain the solution to the larger problem.

7.8 Advanced recursion example

The *Towers of Hanoi* problem is a classic case study in recursion. It involves moving a number of different size disks, stacked on a peg in order of decreasing size, from one tower to another using a third tower as an auxiliary under the constraints that only one disk may be moved at any time and a larger disk can never be on top of a smaller disk. Legend has it that at the creation of the world, the priests of the Temple of Brahma were given this problem with 64 disks and told that when they had completed the task the world would come to an end.

Problem
Move n disks from peg A to peg C, using peg B as needed, according to the following rules:

1. Only one disk may be moved at a time.
2. This disk must be the top disk on a peg.
3. A larger disk can never be placed on top of a smaller disk.

Design

1. How can we define the problem in terms of one or more smaller problems of the same type?

   The key to the problem is not to concentrate our attention on the first step (which must be to move the smallest disk from A to somewhere) but on the hardest step, i.e. moving the bottom disk to peg C.

   There is no way to move the bottom disk until the top $n-1$ disks have been moved. Furthermore they must have been moved to peg B to allow us to move the bottom disk to peg C.

   We then have $n-1$ disks on peg B which must be moved to peg C using peg A (which is free at this moment).

   So the problem "*move n disks from peg A to peg C using peg B*" is equivalent to the following sequence of subproblems:

   - *move* $n-1$ *disks from peg A to peg B using peg C*
   - *move the n'th disk from peg A to peg C.*
   - *move* $n-1$ *disks from peg B to peg C using peg A*

   Notice that the size of the Towers of Hanoi problem is determined by the number of disks involved. This implies that we have redefined the problem in terms of 3 smaller problems of the same type.

2. What instances of the problem can serve as the base cases?

   If $n = 1$ then the problem consists of moving 1 disk from a given peg to another, which we can solve immediately.

3. As the problem size diminishes will we reach the base case?

   Since each call to the function reduces the parameter n by 1, and n is positive, we always reach the base case $n = 1$.

4. How are the solutions from the smaller problem used to build a correct solution to the larger problem?

   When all the three smaller problems are finished the larger problem is completed.

To define a recursive function let Peg be the set of three pegs, $Tower$ be a set of well-defined towers in the sense that any tower consists of three pegs with disks stacked smaller ones on larger ones in some way, and let

$$take\_put: \ Peg \times Peg \times Tower \rightarrow Tower$$

be a function which moves the top disk from one peg to another in a given tower. Then we can define the function which moves n disks from peg_1 to peg_2 using peg_3 in the tower as follows:

$move: \ \mathbf{N} \times Peg \times Peg \times Peg \times Tower \rightarrow Tower$
$move(n, from, to, use, t) =$
  if $(n = 1)$
    then $take\_put(from, to, t)$
    else
      $move(n-1, use, to, from,$
        $take\_put(from, to,$
          $move(n-1, from, use, to, t)))$
  end
end *move*

How long will it take to move the complete tower or for the world to end? To answer this question, let $M(n)$ be a function representing the number of moves the recursive function *move* requires to move an n-disk tower. We define this function recursively.

The base case should be $n = 1$. A single disk is moved directly, i.e. $M(1) = 1$. According to the definition of the function *move* above, we see that

$$M(n) = 2M(n-1) + 1 \qquad n > 1.$$

Therefore the number of moves required is $M(n) = 2^n - 1$.

7.9 A final word

In this section we have presented recursion as a method for defining problems. We have not considered the recursive solution of problems or how recursion is actually implemented on a machine.

Recursion is a very powerful tool for constructing computer routines that otherwise can be quite complex, particularly when the problem is already defined in recursive terms. For such problems recursion can lead to solutions that are much clearer and easier to modify than their iterative counterparts.

However, such recursive definitions do not guarantee that a recursive routine is the best way to solve a problem. Depending on the implementation available, recursion can require a substantial amount of runtime overhead. Thus, the use of recursion illustrates the classical tradeoff between time spent constructing and maintaining a program and the cost in time and memory of executing that program.

We consider the questions of when the use of recursion is appropriate for computing and when it is not to be beyond the scope of this paper and so do not discuss them here.

8 Induction

8.1 Introduction

Scientific discovery often arises from the recognition of a pattern. There are two main aspects of inquiry in science whereby new results can be discovered:

1. *deduction*, and
2. *induction*.

In deduction we accept certain statements as premises and axioms and deduce other statements on the basis of valid inferences.

Induction is the process of discovering general laws by observation and experimentation. In induction we arrive at a conjecture for a general rule by inductive reasoning and prove it by verifying.

8.2 Inductive definitions

Let **S** be an infinite set to be defined. Then an *inductive definition* of **S** consists of the following three components.

1. **Base clause** (or **Basis**) establishes that a finite number of particular objects are elements of the set **S**.
2. **Inductive clause** (or **Induction**) establishes a way to obtain a new element of **S** from some of the previously defined elements.

3. **Extremal clause** asserts that nothing else is in the set **S** other than elements obtained by applying (1) and (2).

The extremal clause can take various forms:

- An object is an element of the set **S** if and only if it can be deduced to be so using only a finite number of applications of the base and inductive steps.
- The set **S** is the smallest set which satisfies the base and inductive steps.
- The set **S** is the set which satisfies the base and inductive steps but which has no proper subset which satisfies them.
- The set **S** is the intersection of all sets which satisfy the properties specified by the base and inductive steps.

Sometimes the extremal clause is left implicit. The recursive definition of a problem (see Section 7.1) is an example of inductive definitions.

Below we give a more precise definition.

Definition (inductive definition)
Let **S** be an infinite set to be defined, and let n and m be finite natural numbers. An *inductive definition* of **S** is a definition with the following three properties:

1. **Base cases:** Objects $b_1, b_2, \ldots, b_n$ are elements of the set **S**, i.e. $b_i \in \mathbf{S}$ for all i such that $1 \leq i \leq n$
2. **Constructor functions:** The functions
   $g_1 : \mathbf{S} \to \mathbf{S}$
   $g_2 : \mathbf{S} \to \mathbf{S}$
   $\vdots$
   $g_m : \mathbf{S} \to \mathbf{S}$
   establish ways to obtain new elements from some of the previously defined elements of the set **S**.
3. **Extremal clause** asserts that nothing else is in the set **S** other than elements obtained by applying (1) and (2).

Examples

- Suppose that the universe is the set **N**. Then the set **E** of even integers can be defined inductively as follows:

  1. Base case: 0 is in **E**, i.e. $0 \in \mathbf{E}$.
  2. Constructor functions: The functions $g_1(n) = n - 2$ and $g_2(n) = n + 2$ both generate even integers. That is, if n is in **E** then $g_1(n) \in \mathbf{E}$ and $g_2(n) \in \mathbf{E}$.
  3. Extremal clause: No integer is in **E** unless it can be shown to be so by a finite number of applications of clauses (1) and (2). We write this symbolically as follows: $\forall e \in \mathbf{E}\ (e = 0 \vee \exists n \in \mathbf{E}(e = g_1(n) \vee e = g_2(n)))$

- Consider the Fibonacci sequence of numbers:

  1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

  The first two members of the sequence are both 1, while each succeeding member is the sum of the two members immediately preceding it.

  Let's define this sequence inductively. It is more convenient if we begin numbering our sequence at 0.

  1. Base cases: $F(0) = 1$ and $F(1) = 1$ are both members of the sequence.
  2. Constructor function: Now we consider the other members. To get a new member (e.g. the (n+1)th member) we define a constructor function as follows:

     $F(n) = F(n-1) + F(n-2)$ for all $n \geq 2$

     This is a recurrence because F is defined in terms of itself.
  3. Extremal clause: All Fibonacci numbers are obtained by applying (1) and (2).

8.3 Proof by induction

The concept of induction also provides powerful techniques for proving assertions of the form '*forall x p(x)*', where $p(x)$ is a predicate and the universe is an inductively defined set (e.g. the set of natural numbers).

A proof by induction consists of two steps which correspond to the base and inductive clauses of the definition of the universe **S** respectively.

1. **Base step** establishes that $p(x)$ is true for every element x specified as a base case in the base clause of the definition of **S**.
2. **Inductive step** establishes that $p(x)$ is true for each element x constructed by the constructor functions of the definition of **S**, assuming that $p(y)$ is true for all elements y used in the construction of x (this is called the *induction hypothesis*).

Since the extremal clause in the definition of **S** guarantees that all elements of **S** are constructed using only the base cases and the constructor functions, $p(x)$ holds for all elements of **S**.

Example
Let **A** be a set which contains only the left and right square brackets, i.e. $\{ [,] \}$ and let **S** be a set of strings over **A** such that

1. Basis: [ ] is an element of **S**.
2. Induction: If x and y are elements of **S**, then

   $[x]$ and xy (i.e. the concatenation of the two strings) are elements of **S**.
3. **S** consists of all strings which can be constructed by a finite number of applications of (1) and (2).

Problem: Let $l(x)$ denote the number of left parentheses in some element x of **S**, and let $r(x)$ denote the number of right parentheses in x.

Prove that $l(x) = r(x)$ for all $x \in \mathbf{S}$.

Proof by induction

1. Basis: $x = [\,]$ in **S**.
   Since $l(x) = 1$ and $r(x) = 1$, $l(x) = r(x)$ holds for the base case.
2. Induction: Assume that $l(x) = r(x)$ and $l(y) = r(y)$ for x and y in **S**. Then
   Constructor 1: Consider $[x]$ which is in **S**.
   $l(\,(x)\,) = l(x) + 1$ and $r(\,(x)\,) = r(x) + 1$
   By induction hypothesis $l(x) = r(x)$, so $l(x) + 1 = r(x) + 1$.
   Hence $l(\,(x)\,) = r(\,(x)\,)$ holds.

   Constructor 2: Consider xy which is in **S**.
   $l(xy) = l(x) + l(y)$ and $r(xy) = r(x) + r(y)$
   By induction hypothesis $l(x) + l(y) = r(x) + r(y)$.
   Hence $l(xy) = r(xy)$ holds.

   Therefore, $l(x) = r(x)$ holds for all x in **S**.

8.4 First principle of mathematical induction

Let $p(n)$ be a predicate on $\mathbf{N}$.

Definition (first principle of mathematical induction)
The *first principle of mathematical induction* is an inference rule of the following form:

if $p(0) \wedge \forall k\ (p(k) \rightarrow p(k+1))$,
then $\forall n\ p(n)$

Proof technique

1. Basis: Show that $p(0)$ is true, using any appropriate proof technique.
2. Induction: Let k be an arbitrary element of $\mathbf{N}$. Assume that $p(k)$ is true (this is the induction hypothesis) and show that $p(k+1)$ is true.

Example
Prove that $S(n) \equiv \sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$
Let $p(n)$ be the predicate stating that the formula is true for n.

Proof by mathematical induction

1. Basis: Since $S(0) = 0^3 = 0$ and $(0(0+1)/2)^2 = 0$, $p(0)$ is true.
2. Induction: Assume that $p(k)$ is true, that is $s(k) \equiv \sum_{i=0}^{k} i^3 = \left(\frac{k(k+1)}{2}\right)^2$.
   Now we show that $p(k+1)$ is true.

   Observe that $S(k+1) = S(k) + (k+1)^3$. Using the inductive hypothesis, we evaluate

$$\begin{aligned} S(k+1) &= S(k) + (k+1)^3 \\ &= (k(k+1)/2)^2 + (k+1)^3 \\ &= (k+1)^2(\,(k/2)^2 + (k+1)) \\ &= (k+1)^2(k^2/4 + k + 1) \\ &= (k+1)^2(k^2 + 4k + 4)/4 \\ &= (k+1)^2(k+2)^2/4 \\ &= ((k+1)(k+2)/2)^2 \end{aligned}$$

We have arrived at $S(k+1) = ((k+1)(k+2)/2)^2$ which shows that $p(k+1)$ is true.

Therefore, the formula holds for any $n \in \mathbf{N}$ according to the first principle of mathematical induction.

8.5 Second principle of mathematical induction

Let $p(n)$ be a predicate on $\mathbf{N}$.

Definition (second principle of mathematical induction)
The *second principle of mathematical induction* is an inference rule of the following form:

if $\forall n\ (\forall k\ (k < n \wedge p(k)) \rightarrow p(n))$,
then $\forall n\ p(n)$

Intuitively, for an arbitrary n if we can show that $p(n)$ is true from the fact that the predicate p is true for all k $(k < n)$ (this includes the base cases also), then we can conclude that the predicate p is true for all n.

Proof technique
Let n be an arbitrary element in $\mathbf{N}$.
Assume that $p(k)$ is true for every $k < n$ and show that $p(n)$ is true.

Example
Any integer $n \geq 2$ can be written as a product of primes.

Proof by mathematical induction
Let $p(n)$ be the predicate 'n can be written as a product of primes'.
Assume that $p(k)$ is true for every $2 \leq k < n$. Then we show that $p(n)$ is true.

The proof is by cases:
Case 1: n is a prime.
Then n is a product of one prime, namely itself, i.e. $p(n)$ is true.

Case 2: n is not a prime.

Then n must have a factor i which satisfies $2 \leq i < n$.
So we can write $n = i \cdot j$ where j also satisfies $2 \leq j < n$
By the induction hypothesis, both i and j can be written as products of primes.
Therefore, n can be written as a product of the products of these primes, i.e. $p(n)$ is true.

In both cases, $p(n)$ is true, so $p(n)$ is true for all $n \geq 2$.

8.6 Set induction

Let *Elem* be a set or a *type*. A set of elements of this type, denoted by *Elem-set*, can be inductively defined as follows:

1. **Base case:** The empty set is a set of type *Elem-set*.
2. **Constructor function:** A function
   $add : Elem \times Elem\text{-}set \rightarrow Elem\text{-}set$
   $add(e, s) = \{e\} \cup s$
   end *add*
   constructs a new set from some previously defined set by adding a given element to it.
3. **Induction axiom:** $(p(\{\}) \wedge ((e \notin \mathbf{S} \wedge p(\mathbf{S})) \Rightarrow p(add(e, \mathbf{S})))) \Rightarrow \forall \mathbf{S} p(\mathbf{S})$
   (where $\mathbf{S}$ is a variable of type *Elem-set*) holds for all predicates p defined on *Elem-set*. Note that the hypothesis $e \notin \mathbf{S}$ in the induction step, which ensures that the element e is not already in the set $\mathbf{S}$, can be assumed without loss of generality because if e is already in the set the implication reduces to $p(\mathbf{S}) \Rightarrow p(\mathbf{S})$ which is automatically true.

Example

The cardinality $|s|$ of a finite set s can be defined as follows:

1. **Base case:** $|\{\}| = 0$.
2. **Induction:** $|add(e, s)| = |s| + 1 \ (e \notin s)$

Prove that $|s_1 \cup s_2| = |s_1| + |s_2| - |s_1 \cap s_2|$ holds for any finite sets s_1 and s_2.

Proof by induction: Let s_1 be an arbitrary fixed set. Then we prove that
$|s_1 \cup s_2| = |s_1| + |s_2| - |s_1 \cap s_2|$ holds for any finite set s_2.

1. *Base case:* $|s_1 \cup \{\}| = |s_1|$ and $|s_1| + |\{\}| - |s_1 \cap \{\}| = |s_1| + |\{\}| - |\{\}| = |s_1| + 0 - 0 = |s_1|$. Hence the equation holds in the base case.

2. *Inductive step:* Let s_2 be a set for which $|s_1 \cup s_2| = |s_1| + |s_2| - |s_1 \cap s_2|$ holds (this is the inductive hypothesis), and let s_2' be a set such that $s_2' = add(e, s_2)$ where $e \notin s_2$, so that $|s_2'| = |s_2| + 1$.

Now consider the expressions $s_1 \cup s_2'$ and $s_1 \cap s_2'$.

$$
\begin{array}{lll}
 & s_1 \cup s_2' & \\
= & s_1 \cup add(e, s_2) & \textit{by substitution} \\
= & s_1 \cup \{e\} \cup s_2 & \textit{by substitution} \\
= & \{e\} \cup s_1 \cup s_2 & \textit{by the commutative law} \\
= & add(e, (s_1 \cup s_2)) & \textit{by substitution}
\end{array}
$$

Similarly,

$$
\begin{array}{lll}
 & s_1 \cap s_2' & \\
= & s_1 \cap add(e, s_2) & \textit{by substitution} \\
= & s_1 \cap (\{e\} \cup s_2) & \textit{by substitution} \\
= & (s_1 \cap \{e\}) \cup (s_1 \cap s_2) & \textit{by the distributive law}
\end{array}
$$

In order to evaluate the last expressions in these two cases, we need to know in the first case whether or not $e \in (s_1 \cup s_2)$ and in the second whether or not $e \in s_1$. However, we know by the induction hypothesis that $e \notin s_2$, so $e \in (s_1 \cup s_2)$ if and only if $e \in s_1$. We consider the two cases separately.

Case 1: Suppose that $e \in s_1$, and hence also $e \in (s_1 \cup s_2)$. In this case the following relations hold:

$$
\begin{array}{l}
s_1 \cap \{e\} = \{e\} \\
add(e, (s_1 \cup s_2)) = s_1 \cup s_2
\end{array}
$$

The first of these implies $s_1 \cap s_2' = add(e, s_1 \cap s_2)$, and since $e \notin s_1 \cap s_2$ because $e \notin s_2$, this in turn implies $|s_1 \cap s_2'| = |s_1 \cap s_2| + 1$

Similarly, the second equation implies $s_1 \cup s_2' = s_1 \cup s_2$, so that $|s_1 \cup s_2'| = |s_1 \cup s_2|$.

So, we can write the following:

$$
\begin{array}{lll}
 & |s_1| + |s_2'| - |s_1 \cap s_2'| & \\
= & |s_1| + (|s_2| + 1) - (|s_1 \cap s_2| + 1) & \textit{by the above} \\
= & |s_1| + |s_2| - |s_1 \cap s_2| & \\
= & |s_1 \cup s_2| & \textit{by the inductive hypothesis} \\
= & |s_1 \cup s_2'| & \textit{by the above}
\end{array}
$$

So the equation holds in this case.

Case 2: Now suppose that $e \notin s_1$, and hence also $e \notin (s_1 \cup s_2)$. In this case, $s_1 \cap \{e\} = \{\ \}$, so, from above, $s_1 \cap s_2^{'} = \{\ \} \cup (s_1 \cap s_2) = s_1 \cap s_2$. Also, $|s_1 \cup s_2^{'}| = |add(e, s_1 \cup s_2)| = |s_1 \cup s_2| + 1$.

So, we can write the following:

$$
\begin{array}{lll}
 & |s_1| + |s_2^{'}| - |s_1 \cap s_2^{'}| & \\
= & |s_1| + (|s_2| + 1) - |s_1 \cap s_2| & \textit{by the above} \\
= & |s_1| + |s_2| - |s_1 \cap s_2| + 1 & \\
= & |s_1 \cup s_2| + 1 & \textit{by the inductive hypothesis} \\
= & |s_1 \cup s_2^{'}| & \textit{by the above}
\end{array}
$$

So in this case we have also arrived at

$$|s_1 \cup s_2^{'}| \quad = \quad |s_1| + |s_2^{'}| - |s_1 \cap s_2^{'}|$$

In other words the equation

$$|s_1 \cup s_2| = |s_1| + |s_2| - |s_1 \cap s_2|$$

is proved.

9 Introduction to propositional modal logic

9.1 Modal formulae

Let $\mathbf{\Phi}$ be a countable set of *atomic formulae* (or propositional variables) and $\mathsf{Fma}(\mathbf{\Phi})$ be the set of all formulae generated from $\mathbf{\Phi}$.

Definition (symbols)
The formulae of modal logic are made up of the following symbols:

- The constant
  $-$ (*falsum* or *bottom*)
- The symbols
  $\Box$ (*box*) and $\rightarrow$ (*implies*)
  ($-$ binds more strongly than $\rightarrow$)

- The atomic formulae letters
  $p, q, r, p_1, q_1, r_1, p_2, q_2, ...$
- The formulae letters $A, B, A_1, B_1, A', B', ...$

Definition (modal formula)
Formulae are formed according to the following rules:

- p is a formula for all $p \in \mathbf{\Phi}$.
- $-$ is a formula.
- If A and B are formulae, i.e. if $A \in \mathsf{Fma}(\mathbf{\Phi})$ and $B \in \mathsf{Fma}(\mathbf{\Phi})$, then

  $$A \to B$$

  is a formula, i.e. $A \to B \in \mathsf{Fma}(\mathbf{\Phi})$.
- If A is a formula, i.e. if $A \in \mathsf{Fma}(\mathbf{\Phi})$, then

  $$\Box A$$

  is a formula.

$\Box A$ can be variously read as:
It is necessarily true that A.
It is known that A.
It will always be true that A.
It ought to be that A.
It is believed that A.
After the program terminates, A.

Note that this definition of formulae is an inductive definition.

Other derived connectives

The following abbreviations define other connectives in terms of the symbols $\Box$ and $\to$ as follows:

Negation:	$\sim A$	is	$A \to -$
Verum (or top):	$\top$	is	$\sim -$
Disjunction:	$A_1 \vee A_2$	is	$\sim A_1 \to A_2$
Conjunction:	$A_1 \wedge A_2$	is	$\sim (A_1 \to \sim A_2)$
Equivalence:	$A_1 \equiv A_2$	is	$(A_1 \to A_2) \wedge (A_2 \to A_1)$
"Diamond":	$\Diamond A$	is	$\sim \Box \sim A$

(see Section 2.1.8 for negation, disjunction and conjunction)

In general, $\Diamond A$ means 'it is sometimes true that A'. For example, $\Diamond A$ means under each of the above reading of $\Box$ as follows:

It is possibly true that A.
It is known that A sometimes.
It will sometimes be true that A.
It ought to be that sometimes A .
It is believed that sometimes A.
There is some execution that terminates with A true.

Examples
The following are all formulae of modal logic:

$$\Box A \to A$$
$$\Box A \to \Box\Box A$$
$$\Diamond -$$
$$\Box A \to \Diamond A$$
$$\Box A \vee \Box \sim A$$
$$\Box(A \to B) \to (\Box A \to \Box B)$$
$$\Diamond A \wedge \Diamond B \to \Diamond(A \wedge B)$$
$$\Box(\Box A \to A) \to \Box A$$

Subformulae

The finite set $\mathsf{Sf}(A)$ of all subformulae of a formula A (i.e. $A \in \mathsf{Fma}(\mathbf{\Phi})$) is defined inductively as follows:

1. $\mathsf{Sf}(p) = \{p\}$
2. $\mathsf{Sf}(-) = \{-\}$
3. $\mathsf{Sf}(A) = \{A_1 \to A_2\} \cup \mathsf{Sf}(A_1) \cup \mathsf{Sf}(A_2)$ if A is $A_1 \to A_2$

9.2 Schemata and substitution

Definition (schema)
A *schema* is a collection of formulae all having a common syntactic form.

For example, by the schema

$$\Box A \to A$$

we mean the collection of formulae

$$\{\Box B \to B \mid B \in \mathsf{Fma}(\mathbf{\Phi})\}$$

The notion of a schema can be made more precise by considering *uniform substitution*, as follows.

Let A and B be any formulae, and let p be an atomic formula.

Definition (uniform substitution)
The *uniform substitution* (or *total substitution*) of B for p in formula A is the procedure of replacing each and every occurrence of p in A by B (see Section 2.1.9).

Definition (substitution instance)
A formula $A^{'}$ is called a *substitution instance* of a formula A if it arises by simultaneous uniform substitution of some formulae for some of the atomic formulae of A.

Thus, if there exist some finitely many atomic formulae $p_1, \ldots, p_n$, and formulae $B_1, \ldots, B_n$, such that $A^{'}$ is the result of simultaneous uniform substitution of B_i for p_i for all $1 \leq i \leq n$ in A, then $A^{'}$ is a *substitution instance* of the formula A.

Let Σ_A be the set of all substitution instances of A. Then a schema may be defined as a set of formulae which is equal to Σ_A for some formula A.

Example
If B is the formula

$$\Box p \to p \text{ where } p \in \mathbf{\Phi}$$

then Σ_B is the set of formulae which was defined above as "the schema $\Box A \to A$".

9.3 Frames and models

Definition (frame)
A *frame* is a pair consisting of a non-empty set **S** and a binary relation R on the set **S**. It is

denoted by $\mathcal{F} = (\mathbf{S}, R)$ where $R \subseteq \mathbf{S} \times \mathbf{S}$.

Examples

- *Time frame:* Let $\mathbf{S}$ be a set of moments of time and let sRt mean 't is later that s'.

  Then $\mathcal{F} = (\mathbf{S}, R)$ is a frame. In this frame $\Box A$ means "at all future times A" and $\Diamond A$ means "at some future time A".

- *Program states:* Let $\mathbf{S}$ be the set of all possible states of a computation process with a program and let sRt mean 'there is an execution of the program that starts in the state s and terminates in the state t'. Then $\Box A$ means "every terminating execution of the program makes A true" and $\Diamond A$ means "there is some execution which terminates with A true"

Definition (model)

A $\mathbf{\Phi}$-*model* or a *model* on a frame $\mathcal{F} = (\mathbf{S}, R)$ is a triple $(\mathbf{S}, R, V)$ where V is a function such that $V : \mathbf{\Phi} \to 2^S$. We denote this by $\mathcal{M} = (\mathbf{S}, R, V)$.

In a model $\mathcal{M} = (\mathbf{S}, R, V)$, the function V assigns to each atomic formula $p \in \mathbf{\Phi}$ a subset $V(p)$ of $\mathbf{S}$. Informally, $V(p)$ can be thought of as the set of points at which p is true.

Definition
The relation '*A is true (or holds) at a point s in a model $\mathcal{M}$*', denoted by

$$\mathcal{M} \models_s A,$$

is defined inductively as follows:

$\mathcal{M} \models_s p$	iff	$s \in V(p)$
$\mathcal{M} \not\models_s -$		(i.e. $-$ is false in any model)
$\mathcal{M} \models_s (A_1 \to A_2)$	iff	$(\mathcal{M} \models_s A_1)$ implies $(\mathcal{M} \models_s A_2)$
$\mathcal{M} \models_s \Box A$	iff	$\forall t \in \mathbf{S}(sRt$ implies $\mathcal{M} \models_t A)$

According to this definition $\Box A$ is true at the point s in a model if and only if A is true at all the points related with the point s in the model.

Examples

(a). Prove that $\mathcal{M} \models_s \sim A$ iff $\mathcal{M} \not\models_s A$ (or $\sim(\mathcal{M} \models_s A)$)

	$\mathcal{M} \models_s \sim A$	
iff	$\mathcal{M} \models_s (A \rightarrow -)$	*by abbreviation*
iff	$\mathcal{M} \models_s A$ implies $\mathcal{M} \models_s -$	*by the definition*
=	$\mathcal{M} \models_s A$ implies *false*	*by definition*
=	$\sim(\mathcal{M} \models_s A)$	
=	$\mathcal{M} \not\models_s A$	

(b). Calculate the truth condition for $A \wedge B$.

	$\mathcal{M} \models_s A \wedge B$	
iff	$\mathcal{M} \models_s \sim(A \rightarrow \sim B)$	*by abbreviation*
iff	$\sim(\mathcal{M} \models_s (A \rightarrow \sim B))$	*using the result of example (a)*
iff	$\sim(\mathcal{M} \models_s A$ implies $\mathcal{M} \models_s \sim B)$	*by the definition*
iff	$\sim(\mathcal{M} \models_s A$ implies $\sim(\mathcal{M} \models_s B))$	*using the result example (a)*
=	$\mathcal{M} \models_s A \wedge \mathcal{M} \models_s B$	*by substitution*

We arrive at

$$\mathcal{M} \models_s A \wedge B \text{ iff } \mathcal{M} \models_s A \wedge \mathcal{M} \models_s B.$$

(c). Prove that $\forall s(\mathcal{M} \models_s \top)$ is true.

	$\mathcal{M} \models_s \top$	
iff	$\mathcal{M} \models_s \sim -$	*by abbreviation*
iff	$\sim(\mathcal{M} \models_s -)$	*by the example (a)*
		$-$ is false in any model, thus
=	*true*	

That is, $\top$ is true in any model.

(d). Work out the truth condition for $\mathcal{M} \models_s (\Box A \rightarrow \Box B)$.

$$
\begin{array}{ll}
 & \mathcal{M} \models_s (\Box A \to \Box B) \\
\text{iff} & (\mathcal{M} \models_s \Box A) \text{ implies } (\mathcal{M} \models_s \Box B) \\
\text{iff} & \forall t \in \mathbf{S} \bullet (sRt \text{ implies } \mathcal{M} \models_t A) \text{ implies } \forall t \in \mathbf{S} \bullet (sRt \text{ implies } \mathcal{M} \models_t B) \\
\text{iff} & \forall t \in \mathbf{S} \bullet (sRt \text{ implies } (\mathcal{M} \models_t A \text{ implies } \mathcal{M} \models_t B)) \\
\text{iff} & \forall t \in \mathbf{S} \bullet (sRt \text{ implies } \mathcal{M} \models_t (A \to B)) \\
\text{iff} & \mathcal{M} \models_s \Box(A \to B)
\end{array}
$$

Here we used the following tautology from propositional logic:

$$(A \to B) \to (A \to C) = (A \to (B \to C)).$$

Thus we arrive at

$$\mathcal{M} \models_s (\Box A \to \Box B) \text{ iff } \mathcal{M} \models_s \Box(A \to B).$$

9.4 Valuation and tautology

Let $\mathcal{M}$ be the $\mathbf{\Phi}$-model $(\mathbf{S}, R, V)$, **Bool** be the set $\{true, false\}$, and let $s \in \mathbf{S}$ be a given point.

Definition (valuation of an atomic formula)
The function $v_s : \mathbf{\Phi} \to \mathbf{Bool}$ defined as

$$v_s(p) = \begin{cases} true & \text{if } s \in V(p) \\ false & \text{otherwise} \end{cases}$$

is called a *valuation* of the atomic formula.

Thus a model on a frame gives rise to a collection $\{v_s \mid s \in \mathbf{S}\}$ of valuations of $\mathbf{\Phi}$, and conversely, such a collection of functions defines the model in which $V(p) = \{s \mid v_s(p) = true\}$.

Definition (quasi-atomic formula)
A formula A is *quasi-atomic* if it is atomic (i.e. $A \in \mathbf{\Phi}$) or if it begins with a $\Box$, i.e. $A = \Box B$ for some $B \in \mathbf{\Phi}$.

If $\mathbf{\Phi}^q$ is the set of all quasi-atomic formulae, then any formula A may be constructed from members of $\mathbf{\Phi}^q \cup \{-\}$ using only the connective $\to$. Hence, any valuation

$$v : \mathbf{\Phi}^q \to \mathbf{Bool}$$

of the quasi-atomic formulae extends uniquely to a valuation

$$v : \mathsf{Fma}(\mathbf{\Phi}) \to \mathbf{Bool}$$

of all formulae.

Definition (tautology)
A formula A is a *tautology* if $v(A) = true$ for every valuation v of its quasi-atomic subformulae.

9.5 Truth and validity

Definition (truth of a formula)
A formula A is *true in a model* $\mathcal{M}$, denoted by $\mathcal{M} \models A$, if it is true at all points in $\mathcal{M}$, i.e. if

$$\forall s \in \mathbf{S}\, (\mathcal{M} \models_s A)$$

Definition (valid)
A formula A is *valid in the frame* $\mathcal{F} = (\mathbf{S}, R)$, denoted by $\mathcal{F} \models A$, if

$$\mathcal{M} \models A \text{ for all models } \mathcal{M} = (\mathbf{S}, R, V) \text{ based on the frame } \mathcal{F}.$$

Definition
If $\mathcal{C}$ is a class of models, then a formula A is *true* in $\mathcal{C}$, denoted by $\mathcal{C} \models A$, if A is true in all members of $\mathcal{C}$.

Definition
If $\mathcal{C}$ is a class of frames, then a formula A is *valid* in $\mathcal{C}$, denoted by $\mathcal{C} \models A$, if A is valid in all members of $\mathcal{C}$.

Definition
A schema is said to be true in a model (respectively, valid in a frame) if all instances of the schema have that property. More generally, we use the notations $\mathcal{M} \models ?$ and $\mathcal{F} \models ?$, where $? \subseteq \mathsf{Fma}(\mathbf{\Phi})$, to mean that all members of $?$ are true in the model $\mathcal{M}$ and valid in the frame $\mathcal{F}$ respectively.

Examples
The following formulae are true in all models, hence valid in all frames.

$$\Box\top$$
$$\Box(A \to B) \to (\Box A \to \Box B)$$
$$\Diamond(A \to B) \to (\Box A \to \Diamond B)$$
$$\Box(A \to B) \to (\Diamond A \to \Diamond B)$$
$$\Box(A \wedge B) \equiv (\Box A \to \Box B)$$
$$\Diamond(A \vee B) \equiv (\Diamond A \to \Diamond B)$$

We prove the first two examples:

Let $\mathcal{M}$ be a model on a frame $\mathcal{F} = (\mathbf{S}, R)$ and let s be an arbitrary point in $\mathbf{S}$.

Proof:

$\mathcal{M} \models_s \Box\top$
iff sRt implies $\mathcal{M} \models_t \top$ *for all* $t \in \mathbf{S}$ *by the definition*; $\mathcal{M} \models_t \top$ *is true in any model, so*
iff sRt implies *true = true* *by the property of implication*

That is, $\mathcal{M} \models_s \Box\top$ is true in any model.

Proof:

$\mathcal{M} \models_s (\Box(A \to B) \to (\Box A \to \Box B))$
iff $(\mathcal{M} \models_s \Box(A \to B))$ implies $(\mathcal{M} \models_s (\Box A \to \Box B))$ *by the definition*
iff $(\mathcal{M} \models_s \Box(A \to B))$ implies $(\mathcal{M} \models_s \Box(A \to B))$ *see Example (d) in Section 9.3*

The last expression is true according to one of the properties of implication.

9.6 Generated submodels

Let $\mathcal{F} = (\mathbf{S}, R)$ be a frame.

Definition (submodel)
If $\mathcal{M} = (\mathbf{S}, R, V)$ is a model and $t \in \mathbf{S}$, then the *submodel of* $\mathcal{M}$ *generated by* t is

$$\mathcal{M}^t = (\mathbf{S}^t, R^t, V^t)$$

where

$$S^t = \{u \in \mathbf{S} \mid tR^*u\}$$
(R^* is the reflexive and transitive closure of the relation R; see Section 6.5)
$$R^t = R \cap (S^t \times S^t)$$
$$V^t(p) = V(p) \cap S^t.$$

Submodel lemma
If $A \in \mathsf{Fma}(\mathbf{\Phi})$, then for any $u \in \mathbf{S}^t$,

$$\mathcal{M}^t \models_u A \qquad \text{iff} \qquad \mathcal{M} \models_u A$$

Proof by induction:
According to the definitions of S^t, R^t and $V^t(p)$, $u \in \mathbf{S}^t$ implies $u \in \mathbf{S}$, sR^tu implies sRu, and $p \in V^t(p)$ implies $p \in V(p)$.

- *Base cases:*

  1. If $A = p$ where $p \in \mathbf{\Phi}$, then

     $\mathcal{M}^t \models_u p$
     iff $p \in V^t(p)$ *by the definition*
     iff $p \in V(p)$ *see definition of* $V^t(p)$
     iff $\mathcal{M} \models_u A$ *by the definition*

  2. If $A = -$, then it is not in any model, thus $\mathcal{M}^t \not\models_u -$ and $\mathcal{M} \not\models_u -$

  Therefore, the assertion is true in both base cases.

- *Inductive step:* Let B and D be formulae such that

  $$\mathcal{M}^t \models_u B \qquad \text{iff} \qquad \mathcal{M} \models_u B$$

  and

  $$\mathcal{M}^t \models_u D \qquad \text{iff} \qquad \mathcal{M} \models_u D$$

  hold.

  1. Let $A = B \to D$. Then

     $\mathcal{M}^t \models_u (B \to D)$
     iff $(\mathcal{M}^t \models_u B) \to (\mathcal{M}^t \models_u D)$ *by the definition*
     iff $(\mathcal{M} \models_u B) \to (\mathcal{M} \models_u D)$ *by the induction hypothesis*
     iff $\mathcal{M} \models_u (B \to D)$ *by the definition*

2. Let $A = \Box B$. Then

	$\mathcal{M}^t \models_u \Box B$	
iff	$uR^t s \to \mathcal{M}^t \models_s B$	*for all* $s \in \mathbf{S}^t$
iff	$uRs \to \mathcal{M} \models_s B$	*by the induction hypothesis*
iff	$\mathcal{M} \models_s \Box B$	*by the definition*

Thus, according to the induction principle the lemma is true.

Corollary

1. $\mathcal{M} \models A$ implies $\mathcal{M}^t \models A$.
2. $\mathcal{M} \models A$ iff A is true in all generated submodels of $\mathcal{M}$.
3. $\mathcal{F} \models A$ iff A is valid in all generated subframes of $\mathcal{F}$.

9.7 p-Morphisms

Let $\mathcal{M}_1 = (\mathbf{S}_1, R_1, V_1)$ and $\mathcal{M}_2 = (\mathbf{S}_2, R_2, V_2)$ be models.

Definition (p-morphism)
A *p-morphism* from $\mathcal{M}_1$ to $\mathcal{M}_2$ is a function $f : \mathbf{S}_1 \to \mathbf{S}_2$ satisfying

sR_1t	implies	$f(s)R_2f(t)$;
$f(s)R_2u$	implies	$\exists t(sR_1t \wedge f(t) = u)$;
$s \in V_1(p)$	iff	$f(s) \in V_2(p)$.

A function satisfying the first two conditions is a p-morphism from the frame $(\mathbf{S}_1, R_1)$ to the frame $(\mathbf{S}_2, R_2)$.

p-Morphism lemma 1

If $A \in \mathsf{Fma}(\mathbf{\Phi})$, then for any $s \in \mathbf{S}_1$,

$$\mathcal{M}_1 \models_s A \text{ iff } \mathcal{M}_2 \models_{f(s)} A.$$

Definition (p-morphic image)
If there is a p-morphism $f : \mathbf{S}_1 \to \mathbf{S}_2$ that is onto (surjective), then the frame $\mathcal{F}_2$ is called a *p-morphic image* of the frame $\mathcal{F}_1$.

p-Morphism lemma 2

If $\mathcal{F}_2$ is a p-morphic image of $\mathcal{F}_1$, then for any formula A,

$$\mathcal{F}_1 \models A \text{ implies } \mathcal{F}_2 \models A.$$

Proofs of these lemmas are as exercises left for the reader.

10 Conclusion

In this report we have presented material for a course on mathematics covering those topics which are the most important for those wishing to study formal methods, such as RAISE, VDM, and Z. This material could be used immediately prior to a course on formal methods as a short introductory course on mathematics for students without the required mathematical knowledge. It could also, suitably extended, form the bulk of a longer course on "Mathematics for computer science" forming part of the curriculum in computer science departments in universities, and thus could be useful to students of computer science in general.

We have also presented in the final section of the report material for an introductory short course on modal logic which could similarly be presented before a course on Duration Calculus.

Teaching materials for each of the individual sections included in this report, specifically overhead projector foils for lecturers, are available from UNU/IIST and can in fact be downloaded electronically from UNU/IIST's home pages at the following URL:
http://www.iist.unu.edu/home/Unuiist/newrh/II/1/3/2/page.html.

11 Acknowledgements

Many thanks to my supervisor Richard Moore for reading and revising both the manuscript and the final version of this report, and providing helpful comments on all sections of the report. The text and its contents became more readable and contain fewer errors thanks to him. My thanks to Dang Van Hung for reading and making valuable comments on the section on modal logic. Thanks also to all at UNU/IIST who helped with this project and who made my stay at UNU/IIST possible.

References

[1] Allen B. Ticker. Computing curricula 1991. Communications of the ACM, p. 68–84, June 1991.

[2] The RAISE Method Group. The RAISE Development Method. BCS Practitioner Series. Prentice Hall, 1995.

[3] The RAISE Language Group. The RAISE Specification Language. BCS Practitioner Series. Prentice Hall, 1992.

[4] Z. Manna, R. Waldinger. The Logical Basis for Computer Programming. Volume 1: Deductive Reasoning. Addison-Wesley Publishing Company,1985.

[5] R. Goldblatt. Logics of Time and Computation. Second Edition. CSLI, 1992.

[6] Juan C. Bicarregui, etc. Proof in VDM: A Practitioner's Guide. Springer-Verlag. 1994

[7] K. Sugihara. Discrete Mathematics for Computer Science. Department of Information and Computer Science, College of Natural Science, University of Hawaii.
http://www.ics.hawaii.edu/ sugihara/course/

[8] S. Ramaswamy. Discrete Structures. Department of Computer Science, Tennessee Technological University.
http://www.csc.tntech.edu/ srini/DM/

[9] Recursion Tutorial. Web site. Information Technology Centre, National University of Ireland, Galway.
http://www.it.ucg/CAI_Tutor/

Index