COMPP2111 Week 1

Term 1, 2019

Discrete Mathematics Recap

Summary of topics

- Sets
- Formal languages
- Relations
- Functions
- Propositional Logic

Summary of topics

- Sets
- Formal languages
- Relations
- Functions
- Propositional Logic

Sets

A set is defined by the collection of its elements.
Sets are typically described by:
(a) Explicit enumeration of their elements

$$
\begin{aligned}
S_1 &= \{a, b, c\} = \{a, a, b, b, b, c\} \\
&= \{b, c, a\} = \ldots \quad \text{three elements} \\
S_2 &= \{a, \{a\}\} \quad \text{two elements} \\
S_3 &= \{a, b, \{a, b\}\} \quad \text{three elements} \\
S_4 &= \{\} \quad \text{zero elements} \\
S_5 &= \{\{\{\}\}\} \quad \text{one element} \\
S_6 &= \{\,\{\}, \{\{\}\}\,\} \quad \text{two elements}
\end{aligned}
$$

(b) Specifying the properties their elements must satisfy; the elements are taken from some 'universal' domain, $\mathcal{U}$. A typical description involves a **logical** property $P(x)$

$$S = \{\, x : x \in \mathcal{U} \text{ and } P(x) \,\} = \{\, x \in \mathcal{U} : P(x) \,\}$$

We distinguish between an element and the set comprising this single element. Thus always $a \neq \{a\}$.
Set $\{\}$ is empty (no elements);
set $\{\{\}\}$ is nonempty — it has one element.
There is only one empty set; only one set consisting of a single a; only one set of all natural numbers.

(c) Constructions from other sets (already defined)

- Union, intersection, set difference, symmetric difference, complement
- **Power set** $\mathrm{Pow}(X) = \{ A : A \subseteq X \}$
- Cartesian product (below)
- Empty set $\emptyset$
  $\emptyset \subseteq X$ for all sets X.

$S \subseteq T$ — S is a **subset** of T; includes the case of $T \subseteq T$
$S \subset T$ — a **proper** subset: $S \subseteq T$ and $S \neq T$

NB

An element of a set and a subset of that set are two different concepts

$$a \in \{a, b\}, \quad a \not\subseteq \{a, b\}; \qquad \{a\} \subseteq \{a, b\}, \quad \{a\} \notin \{a, b\}$$

Cardinality

Number of elements in a set X (various notations):

$$|X| = \#(X) = \mathrm{card}(X)$$

Fact

Always $|Pow(X)| = 2^{|X|}$

$|\emptyset| = 0 \qquad \mathrm{Pow}(\emptyset) = \{\emptyset\} \qquad |\mathrm{Pow}(\emptyset)| = 1$
$\mathrm{Pow}(\mathrm{Pow}(\emptyset)) = \{\emptyset, \{\emptyset\}\} \qquad |\mathrm{Pow}(\mathrm{Pow}(\emptyset))| = 2 \quad \ldots$

$|\{a\}| = 1 \qquad \mathrm{Pow}(\{a\}) = \{\emptyset, \{a\}\} \qquad |\mathrm{Pow}(\{a\})| = 2 \quad \ldots$

$[m, n]$ — interval of integers; it is empty if $n < m$
$|[m, n]| = n - m + 1$, for $n \geq m$

Sets of Numbers

Natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$
Positive integers $\{1, 2, \ldots\}$
Common notation $\mathbb{N}_{>0} = \mathbb{Z}_{>0} = \mathbb{N} \setminus \{0\}$

Integers $\mathbb{Z} = \{\ldots, -n, -(n-1), \ldots, -1, 0, 1, 2, \ldots\}$
Rational numbers (fractions) $\mathbb{Q} = \left\{ \frac{m}{n} : m, n \in \mathbb{Z}, n \neq 0 \right\}$
Real numbers (decimal or binary expansions) $\mathbb{R}$
$r = a_1 a_2 \ldots a_k \,.\, b_1 b_2 \ldots$

Intervals of numbers (applies to any type)

$$[a, b] = \{x | a \leq x \leq b\}; \quad (a, b) = \{x | a < x < b\}$$

$$[a, b] \supseteq [a, b), \ (a, b] \supseteq (a, b)$$

NB

$(a, a) = (a, a] = [a, a) = \emptyset$; *however* $[a, a] = \{a\}$.

Intervals of $\mathbb{N}, \mathbb{Z}$ are finite: if $m \leq n$

$$[m, n] = \{m, m+1, \dots, n\} \qquad |[m, n]| = n - m + 1$$

Set Operations

Union $A \cup B$; Intersection $A \cap B$

Note that there is a correspondence between set operations and logical operators (to be discussed later)

We say that A, B are **disjoint** if $A \cap B = \emptyset$

NB

$A \cup B = B \leftrightarrow A \subseteq B \qquad A \cap B = B \leftrightarrow A \supseteq B$

Other set operations

- $A \setminus B$ — **difference**, set difference, relative complement
  It corresponds (logically) to a but not b
- $A \oplus B$ — **symmetric difference**

$$A \oplus B \stackrel{\text{def}}{=} (A \setminus B) \cup (B \setminus A)$$

  It corresponds to a and not b or b and not a; also known as **xor (exclusive or)**
- A^c — set **complement** w.r.t. the 'universe' $\mathcal{U}$
  It corresponds to 'not a'

Laws of Set Operations

Commutativity	$A \cup B = B \cup A$
	$A \cap B = B \cap A$
Associativity	$(A \cup B) \cup C = A \cup (B \cup C)$
	$(A \cap B) \cap C = A \cap (B \cap C)$
Distribution	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
Identity	$A \cup \emptyset = A$
	$A \cap \mathcal{U} = A$
Complementation	$A \cup (A^c) = \mathcal{U}$
	$A \cap (A^c) = \emptyset$

Other useful set laws

The following are all derivable from the previous 10 laws.

Idempotence	$A \cap A = A$
	$A \cup A = A$
Double complementation	$(A^c)^c = A$
Annihilation	$A \cap \emptyset = \emptyset$
	$A \cup \mathcal{U} = \mathcal{U}$
de Morgan's Laws	$(A \cap B)^c = A^c \cup B^c$
	$(A \cup B)^c = A^c \cap B^c$

Example (Idempotence of $\cup$)

$$
\begin{aligned}
A &= A \cup \emptyset && \text{(Identity)} \\
&= A \cup (A \cap A^c) && \text{(Complementation)} \\
&= (A \cup A) \cap (A \cup A^c) && \text{(Distributivity)} \\
&= (A \cup A) \cap \mathcal{U} && \text{(Complementation)} \\
&= (A \cup A) && \text{(Identity)}
\end{aligned}
$$

Example (Idempotence of $\cup$)

$$
\begin{aligned}
A &= A \cup \emptyset && \text{(Identity)}\\
&= A \cup (A \cap A^c) && \text{(Complementation)}\\
&= (A \cup A) \cap (A \cup A^c) && \text{(Distributivity)}\\
&= (A \cup A) \cap \mathcal{U} && \text{(Complementation)}\\
&= (A \cup A) && \text{(Identity)}
\end{aligned}
$$

Example (Idempotence of $\cup$)

$$
\begin{aligned}
A &= A \cup \emptyset && \text{(Identity)}\\
&= A \cup (A \cap A^c) && \text{(Complementation)}\\
&= (A \cup A) \cap (A \cup A^c) && \text{(Distributivity)}\\
&= (A \cup A) \cap \mathcal{U} && \text{(Complementation)}\\
&= (A \cup A) && \text{(Identity)}
\end{aligned}
$$

Example (Idempotence of $\cup$)

$$\begin{aligned}
A &= A \cup \emptyset && \text{(Identity)} \\
&= A \cup (A \cap A^c) && \text{(Complementation)} \\
&= (A \cup A) \cap (A \cup A^c) && \text{(Distributivity)} \\
&= (A \cup A) \cap \mathcal{U} && \text{(Complementation)} \\
&= (A \cup A) && \text{(Identity)}
\end{aligned}$$

Example (Idempotence of $\cup$)

$$\begin{aligned}
A &= A \cup \emptyset && \text{(Identity)}\\
&= A \cup (A \cap A^c) && \text{(Complementation)}\\
&= (A \cup A) \cap (A \cup A^c) && \text{(Distributivity)}\\
&= (A \cup A) \cap \mathcal{U} && \text{(Complementation)}\\
&= (A \cup A) && \text{(Identity)}
\end{aligned}$$

A useful result

Definition

If A is a set defined using $\cap$, $\cup$, $\emptyset$ and $\mathcal{U}$, then dual(A) is the expression obtained by replacing $\cap$ with $\cup$ (and vice-versa) and $\emptyset$ with $\mathcal{U}$ (and vice-versa).

Theorem (Principle of Duality)

If you can prove $A_1 = A_2$ using the Laws of Set Operations then you can prove $dual(A_1) = dual(A_2)$

Example

Absorption law: $A \cup (A \cap B) = A$

Dual: $A \cap (A \cup B) = A$

Application (Idempotence of $\cap$)

Recall Idempotence of $\cup$:

$$
\begin{aligned}
A &= A \cup \emptyset && \text{(Identity)} \\
&= A \cup (A \cap A^c) && \text{(Complementation)} \\
&= (A \cup A) \cap (A \cup A^c) && \text{(Distributivity)} \\
&= (A \cup A) \cap \mathcal{U} && \text{(Complementation)} \\
&= (A \cup A) && \text{(Identity)}
\end{aligned}
$$

Application (Idempotence of $\cap$)

Invoke the dual laws!

$$\begin{aligned}
A &= A \cap \mathcal{U} && \text{(Identity)} \\
&= A \cap (A \cup A^c) && \text{(Complementation)} \\
&= (A \cap A) \cup (A \cap A^c) && \text{(Distributivity)} \\
&= (A \cap A) \cup \emptyset && \text{(Complementation)} \\
&= (A \cap A) && \text{(Identity)}
\end{aligned}$$

Cartesian Product

$S \times T \stackrel{\text{def}}{=} \{\ (s,t) : s \in S,\ \ t \in T\ \}$ where (s,t) is an **ordered** pair

$\bigtimes_{i=1}^{n} S_i \stackrel{\text{def}}{=} \{\ (s_1, \ldots, s_n) : s_k \in S_k, \text{ for } 1 \leq k \leq n\ \}$

$S^2 = S \times S, \quad S^3 = S \times S \times S, \ldots, \quad S^n = \times_1^n S, \ldots$

$\emptyset \times S = \emptyset$, for every S
$|S \times T| = |S| \cdot |T|, \quad |\times_{i=1}^{n} S_i| = \prod_{i=1}^{n} |S_i|$

Summary of topics

- Sets
- Formal languages
- Relations
- Functions
- Propositional Logic

Formal Languages

Σ — **alphabet**, a finite, nonempty set

Examples (of various alphabets and their intended uses)

$\Sigma = \{a, b, \ldots, z\}$ for single words (in lower case)
$\Sigma = \{\sqcup, -, a, b, \ldots, z\}$ for composite terms
$\Sigma = \{0, 1\}$ for binary integers
$\Sigma = \{0, 1, \ldots, 9\}$ for decimal integers

The above cases all have a natural ordering; this is not required in general, thus the set of all Chinese characters forms a (formal) alphabet.

Definition

word — any finite string of symbols from Σ
empty word — λ (sometimes ϵ)

Example

$w = aba$, $w = 01101\ldots1$, etc.

length(w) — # of symbols in w
length(aaa) $= 3$, length(λ) $= 0$
The only operation on words (discussed here) is **concatenation**, written as juxtaposition $vw, wvw, abw, wbv, \ldots$

NB

$\lambda w = w = w\lambda$
length(vw) = length(v) + length(w)

Notation: Σ^k — set of all words of length k
We often identify $\Sigma^0 = \{\lambda\}$, $\Sigma^1 = \Sigma$
Σ^* — set of all words (of all [finite] lengths)
Σ^+ — set of all nonempty words (of any positive length)

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots; \quad \Sigma^{\leq n} = \bigcup_{i=0}^{n} \Sigma^i$$

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots = \Sigma^* \setminus \{\lambda\}$$

A **language** is a subset of Σ^*. Typically, only the subsets that can be formed (or described) according to certain rules are of interest. Such a collection of 'descriptive/formative' rules is called a **grammar**.

Examples: Programming languages, Database query languages

Example (Decimal numbers)

The “language” of all numbers written in decimal to at most two decimal places can be described as follows:

- $\Sigma = \{-, ., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Consider all words $w \in \Sigma^*$ which satisfy the following:
    - w contains at most one instance of $-$, and if it contains an instance then it is the first symbol.
    - w contains at most one instance of ., and if it contains an instance then it is preceeded by a symbol in $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and followed by either one or two symbols in that set.
    - w contains at least one symbol from $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

NB

According to these rules 123, 123.0 *and* 123.00 *are all (distinct) words in this language.*

Example (HTML documents)

Take $\Sigma = \{$“$<$html$>$”, “$<$/html$>$”, “$<$head$>$”, “$<$/head$>$”, “$<$body$>$”, $\ldots\}$.

The (language of) **valid HTML documents** is loosely described as follows:

- Starts with “$<$html$>$”
- Next symbol is “$<$head$>$”
- Followed by zero or more symbols from the set of HeadItems (defined elsewhere)
- Followed by “$<$/head$>$”
- Followed by “$<$body$>$”
- Followed by zero or more symbols from the set of BodyItems (defined elsewhere)
- Followed by “$<$/body$>$”
- Followed by “$<$/html$>$”

Set Operations for Languages

Languages are sets, so the standard set operations ($\cap$, $\cup$, $\setminus$, $\oplus$, etc) can be used to build new languages.

Two set operations that apply to languages uniquely:

- Concatenation (written as juxtaposition):
  $XY = \{xy \ : \ x \in X \text{ and } y \in Y\}$
- Kleene star: X^* is the set of words that are made up by concatenating 0 or more words in X

Set Operations for Languages

Example

Let $A = \{aa, bb\}$ and $B = \{\lambda, c\}$ be languages over $\Sigma = \{a, b, c\}$.

- $A \cup B = \{\lambda, c, aa, bb\}$
- $AB = \{aa, bb, aac, bbc\}$
- $AA = \{aaaa, aabb, bbaa, bbbb\}$
- $A^* = \{\lambda, aa, bb, aaaa, aabb, bbaa, bbbb, aaaaaa, \ldots\}$
- $B^* = \{\lambda, c, cc, ccc, cccc, \ldots\}$
- $\{\lambda\}^* =$
- $\emptyset^* =$

Set Operations for Languages

Example

Let $A = \{aa, bb\}$ and $B = \{\lambda, c\}$ be languages over $\Sigma = \{a, b, c\}$.

- $A \cup B = \{\lambda, c, aa, bb\}$
- $AB = \{aa, bb, aac, bbc\}$
- $AA = \{aaaa, aabb, bbaa, bbbb\}$
- $A^* = \{\lambda, aa, bb, aaaa, aabb, bbaa, bbbb, aaaaaa, \ldots\}$
- $B^* = \{\lambda, c, cc, ccc, cccc, \ldots\}$
- $\{\lambda\}^* =$
- $\emptyset^* =$

Set Operations for Languages

Example

Let $A = \{aa, bb\}$ and $B = \{\lambda, c\}$ be languages over $\Sigma = \{a, b, c\}$.

- $A \cup B = \{\lambda, c, aa, bb\}$
- $AB = \{aa, bb, aac, bbc\}$
- $AA = \{aaaa, aabb, bbaa, bbbb\}$
- $A^* = \{\lambda, aa, bb, aaaa, aabb, bbaa, bbbb, aaaaaa, \ldots\}$
- $B^* = \{\lambda, c, cc, ccc, cccc, \ldots\}$
- $\{\lambda\}^* = \{\lambda\}$
- $\emptyset^* =$

Set Operations for Languages

Example

Let $A = \{aa, bb\}$ and $B = \{\lambda, c\}$ be languages over $\Sigma = \{a, b, c\}$.

- $A \cup B = \{\lambda, c, aa, bb\}$
- $AB = \{aa, bb, aac, bbc\}$
- $AA = \{aaaa, aabb, bbaa, bbbb\}$
- $A^* = \{\lambda, aa, bb, aaaa, aabb, bbaa, bbbb, aaaaaa, \ldots\}$
- $B^* = \{\lambda, c, cc, ccc, cccc, \ldots\}$
- $\{\lambda\}^* = \{\lambda\}$
- $\emptyset^* =$

Set Operations for Languages

Example

Let $A = \{aa, bb\}$ and $B = \{\lambda, c\}$ be languages over $\Sigma = \{a, b, c\}$.

- $A \cup B = \{\lambda, c, aa, bb\}$
- $AB = \{aa, bb, aac, bbc\}$
- $AA = \{aaaa, aabb, bbaa, bbbb\}$
- $A^* = \{\lambda, aa, bb, aaaa, aabb, bbaa, bbbb, aaaaaa, \ldots\}$
- $B^* = \{\lambda, c, cc, ccc, cccc, \ldots\}$
- $\{\lambda\}^* = \{\lambda\}$
- $\emptyset^* = \{\lambda\}$

Set Operations for Languages

Example

Let $A = \{aa, bb\}$ and $B = \{\lambda, c\}$ be languages over $\Sigma = \{a, b, c\}$.

- $A \cup B = \{\lambda, c, aa, bb\}$
- $AB = \{aa, bb, aac, bbc\}$
- $AA = \{aaaa, aabb, bbaa, bbbb\}$
- $A^* = \{\lambda, aa, bb, aaaa, aabb, bbaa, bbbb, aaaaaa, \ldots\}$
- $B^* = \{\lambda, c, cc, ccc, cccc, \ldots\}$
- $\{\lambda\}^* = \{\lambda\}$
- $\emptyset^* = \{\lambda\}$

Summary of topics

- Sets
- Formal languages
- Relations
- Functions
- Propositional Logic

Relations and Functions

Relations are an abstraction used to capture the idea that the objects from certain domains (often the same domain for several objects) are *related*. These objects may

- influence one another (each other for binary relations; self(?) for unary)
- share some common properties
- correspond to each other precisely when some constraints are satisfied

Functions capture the idea of transforming *inputs* into *outputs*.

In general, functions and relations formalise the concept of interaction among objects from various domains; however, there must be a specified domain for each type of objects.

Applications

Relations and functions are ubiquitous in Computer Science

- Databases are collections of relations
- Common data structures (e.g. graphs) are relations
- Any ordering is a relation
- Functions/procedures/programs compute relations between their input and output

Relations are therefore used in most problem specifications and to describe formal properties of programs.
For this reason, studying relations and their properties helps with formalisation, implementation and verification of programs.

Relations

An **n-ary relation** is a subset of the cartesian product of n sets.

$$R \subseteq S_1 \times S_2 \times \ldots \times S_n$$

$$x \in R \rightarrow x = (x_1, x_2, \ldots x_n) \text{ where each } x_i \in S_i$$

If $n = 2$ we have a **binary** relation $R \subseteq S \times T$.
(mostly we consider binary relations)
equivalent notations: $(x_1, x_2, \ldots x_n) \in R \iff R(x_1, x_2, \ldots x_n)$
for binary relations: $(x, y) \in R \iff R(x, y) \iff xRy$.

Examples

- Equality: $=$
- Inequality: $\leq$, $\geq$, $<$, $>$, $\neq$
- Divides relation: $|$ (recall $m|n$ if $n = km$ for some $k \in \mathbb{Z}$)
- Element of: $\in$
- Subset, superset: $\subseteq$, $\subset$, $\supseteq$, $\supset$
- Size functions (sort of): $|\cdot|$, $\mathrm{length}(\cdot)$

Database Examples

Example (Course enrolments)

S = set of CSE students
(S can be a subset of the set of all students)
C = set of CSE courses
(likewise)
E = enrolments = $\{ (s, c) : s \text{ takes } c \}$

$$E \subseteq S \times C$$

In practice, almost always there are various 'onto' (nonemptiness) and 1–1 (uniqueness) constraints on database relations.

Example (Class schedule)

C = CSE courses
T = starting time (hour & day)
R = lecture rooms
S = schedule =

$$\{\,(c,t,r) : c \text{ is at } t \text{ in } r\,\} \subseteq C \times T \times R$$

Example (sport stats)

$$R \subseteq \text{competitions} \times \text{results} \times \text{years} \times \text{athletes}$$

n-ary Relations

Relations can be defined linking $k \geq 1$ domains $D_1, \dots, D_k$ simultaneously.
In database situations one also allows for *unary* $(n = 1)$ relations.
Most common are **binary** relations

$$R \subseteq S \times T; \quad R = \{(s,t) \ : \ \text{"some property that links } s,t\text{"}\}$$

For related s, t we can write $(s,t) \in R$ or sRt; for unrelated items either $(s,t) \notin R$ or $s \not R t$.
R can be defined by

- explicit enumeration of interrelated k-tuples (ordered pairs in case of binary relations);
- properties that identify relevant tuples within the entire $D_1 \times D_2 \times \ldots \times D_k$;
- construction from other relations.

Relation R as Correspondence From S to T

Given $R \subseteq S \times T$, $A \subseteq S$, and $B \subseteq T$.

- $R(A) \stackrel{\text{def}}{=} \{t \in T \;:\; (s,t) \in R \text{ for some } s \in A\}$
- Converse relation $R^{\leftarrow} \subseteq T \times S$:

$$R^{\leftarrow} \stackrel{\text{def}}{=} \{(t,s) \in T \times S \;:\; (s,t) \in R\}$$

- $R^{\leftarrow}(B) = \{s \in S \;:\; (s,t) \in R \text{ for some } t \in B\}$

Observe that $(R^{\leftarrow})^{\leftarrow} = R$.

Binary Relations

A binary relation, say $R \subseteq S \times T$, can be presented as a matrix with rows enumerated by (the elements of) S and the columns by T; eg. for $S = \{s_1, s_2, s_3\}$ and $T = \{t_1, t_2, t_3, t_4\}$ we may have

$$\begin{bmatrix} \bullet & \circ & \bullet & \bullet \\ \circ & \bullet & \bullet & \bullet \\ \bullet & \bullet & \circ & \circ \end{bmatrix}$$

Relations on a Single Domain

Particularly important are binary relationships between the elements of the same set. We say that 'R is a relation on S' if

$$R \subseteq S \times S$$

Such relations can be visualized as a directed graph:

- Vertices: Elements of S
- Edges: Elements of R

Example

$S = \{1, 2, 3\}$
$R = \{(1, 2), (2, 3), (3, 2)\}$

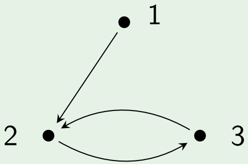
As a matrix:

$$\begin{pmatrix} \circ & \bullet & \circ \\ \circ & \circ & \bullet \\ \circ & \bullet & \circ \end{pmatrix}$$

Example

$S = \{1, 2, 3\}$
$R = \{(1, 2), (2, 3), (3, 2)\}$

As a graph:



Special (Trivial) Relations

(all w.r.t. set S)

Identity (diagonal, equality) $E = \{\ (x, x) : x \in S\ \}$

Empty $\emptyset$

Universal $\mathcal{U} = S \times S$

Important Properties of Binary Relations $R \subseteq S \times S$

(R)	reflexive	$(x, x) \in R$	$\forall x \in S$
(AR)	antireflexive	$(x, x) \notin R$	$\forall x \in S$
(S)	symmetric	$(x, y) \in R \to (y, x) \in R$	$\forall x, y \in S$
(AS)	antisymmetric	$(x, y), (y, x) \in R \to x = y$	$\forall x, y \in S$
(T)	transitive	$(x, y), (y, z) \in R \to (x, z) \in R$	$\forall x, y, z \in S$

NB

An object, notion etc. is considered to satisfy a property if none of its instances violates any defining statement of that property.

Examples

(R) reflexive $(x, x) \in R$ for all $x \in S$ $\begin{bmatrix} \bullet & \bullet & \circ \\ \circ & \bullet & \circ \\ \bullet & \circ & \bullet \end{bmatrix}$

(AR) antireflexive $(x, x) \notin R$ $\begin{bmatrix} \circ & \bullet & \bullet \\ \circ & \circ & \circ \\ \bullet & \circ & \circ \end{bmatrix}$

(S) symmetric $(x, y) \in R \to (y, x) \in R$ $\begin{bmatrix} \bullet & \circ & \bullet \\ \circ & \circ & \bullet \\ \bullet & \bullet & \circ \end{bmatrix}$

(AS) antisymmetric $(x, y), (y, x) \in R \to x = y$
$\begin{bmatrix} \bullet & \bullet & \circ \\ \circ & \circ & \bullet \\ \bullet & \circ & \circ \end{bmatrix}$

(T) transitive $(x, y), (y, z) \in R \to (x, z) \in R$
$\begin{bmatrix} \circ & \circ & \bullet \\ \bullet & \bullet & \bullet \\ \circ & \circ & \circ \end{bmatrix}$

Common relations and their properties

	(R)	(AR)	(S)	(AS)	(T)
$=$	✓		✓	✓	✓
$\leq$	✓			✓	✓
$<$		✓		✓	✓
$\emptyset$		✓	✓	✓	✓
$\mathcal{U}$	✓		✓		✓
$\mid$	✓			✓	✓

Common relations and their properties

	(R)	(AR)	(S)	(AS)	(T)
$=$	✓		✓	✓	✓
$\leq$	✓			✓	✓
$<$		✓		✓	✓
$\emptyset$		✓	✓	✓	✓
$\mathcal{U}$	✓		✓		✓
$\mid$	✓			✓	✓

Common relations and their properties

	(R)	(AR)	(S)	(AS)	(T)
$=$	✓		✓	✓	✓
$\leq$	✓			✓	✓
$<$					
$\emptyset$					
$\mathcal{U}$					
$\mid$					

Common relations and their properties

	(R)	(AR)	(S)	(AS)	(T)
$=$	✓		✓	✓	✓
$\leq$	✓			✓	✓
$<$		✓		✓	✓
$\emptyset$		✓	✓	✓	✓
$\mathcal{U}$	✓		✓		✓
$\mid$	✓			✓	✓

Common relations and their properties

	(R)	(AR)	(S)	(AS)	(T)
$=$	✓		✓	✓	✓
$\leq$	✓			✓	✓
$<$		✓		✓	✓
$\emptyset$		✓	✓	✓	✓
$\mathcal{U}$	✓		✓		✓
$\mid$	✓			✓	✓

Common relations and their properties

	(R)	(AR)	(S)	(AS)	(T)
$=$	✓		✓	✓	✓
$\leq$	✓			✓	✓
$<$		✓		✓	✓
$\emptyset$		✓	✓	✓	✓
$\mathcal{U}$	✓		✓		✓
$\mid$	✓			✓	✓

Common relations and their properties

	(R)	(AR)	(S)	(AS)	(T)
$=$	✓		✓	✓	✓
$\leq$	✓			✓	✓
$<$		✓		✓	✓
$\emptyset$		✓	✓	✓	✓
$\mathcal{U}$	✓		✓		✓
$\mid$	✓			✓	✓

Interaction of Properties

A relation *can* be both symmetric and antisymmetric. Namely, when R consists only of some pairs $(x, x), x \in S$.
A relation *cannot* be simultaneously reflexive and antireflexive (unless $S = \emptyset$).

NB

$$\left.\begin{array}{r}\textit{nonreflexive}\\ \textit{nonsymmetric}\end{array}\right\}\ \textit{is not the same as}\ \left\{\begin{array}{l}\textit{antireflexive/irreflexive}\\ \textit{antisymmetric}\end{array}\right.$$

Equivalence Relations and Partitions

Relation R is called an *equivalence* relation if it satisfies (R), (S), (T). Every equivalence R defines *equivalence classes* on its domain S.

The equivalence class $[s]$ (w.r.t. R) of an element $s \in S$ is

$$[s] = \{ \, t \in S : tRs \, \}$$

This notion is well defined only for R which is an equivalence relation. Collection of all equivalence classes $[S]_R = \{ \, [s] : s \in S \, \}$ is a partition of S

$$S = \bigcup_{s \in S} [s]$$

Thus the equivalence classes are disjoint and jointly cover the entire domain. It means that every element belongs to one (and only one) equivalence class.

We call $s_1, s_2, \ldots$ *representatives* of (different) equivalence classes

For $s, t \in S$ either $[s] = [t]$, when sRt, or $[s] \cap [t] = \emptyset$, when $s \not R t$.

We commonly write $s \sim_R t$ when s, t are in the same equivalence class.

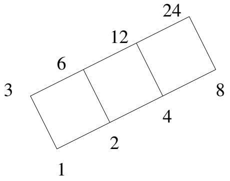
In the opposite direction, a partition of a set defines the equivalence relation on that set. If $S = S_1 \dot\cup \ldots \dot\cup S_k$, then we specify $s \sim t$ exactly when s and t belong to the same S_i.

Partial Order

A **partial order** $\preceq$ on S satisfies (R), (AS), (T).
We call $(S, \preceq)$ a **poset** — partially ordered set

Every finite poset can be represented as a **Hasse diagram**, where a line is drawn *upward* from x to y if $x \prec y$ and there is no z such that $x \prec z \prec y$

Hasse diagram for positive divisors of 24



$p \preceq q$ if, and only if, $p \mid q$

Ordering Concepts

- *Minimal* and *maximal* elements (they always exist in every finite poset)
- *Minimum* and *maximum* — unique minimal and maximal element (might not exist)
- *lub* (least upper bound) and *glb* (greatest lower bound) of a subset $A \subseteq S$ of elements
  $\text{lub}(A)$ — minimum of $\{x \in S \text{ s.t. } x \succeq a \text{ for all } a \in A\}$
  $\text{glb}(A)$ — maximum of $\{x \in S \text{ s.t. } x \preceq a \text{ for all } a \in A\}$

Examples

- $\mathrm{Pow}(\{a,b,c\})$ with the order $\subseteq$
  $\emptyset$ is minimum; $\{a,b,c\}$ is maximum
- $\mathrm{Pow}(\{a,b,c\}) \setminus \{\{a,b,c\}\}$ (proper subsets of $\{a,b,c\}$)
  Each two-element subset $\{a,b\}, \{a,c\}, \{b,c\}$ is maximal.
  - But there is no maximum
- $\{1,2,3,4,6,8,12,24\}$ partially ordered by divisibility
  - e.g. $\mathrm{lub}(\{4,6\}) = 12$; $\mathrm{glb}(\{4,6\}) = 2$
- $\{1,2,3\}$ partially ordered by divisibility
  - $\{2,3\}$ has no lub
- $\{2,3,6\}$ partially ordered by divisibility
  - $\{2,3\}$ has no glb
- $\{1,2,3,12,18,36\}$ partially ordered by divisibility
  - $\{2,3\}$ has no lub ($12, 18$ are minimal upper bounds)

Summary of topics

- Sets
- Formal languages
- Relations
- Functions
- Propositional Logic

Functions

A **function**, $f : S \to T$, is a binary relation $f \subseteq S \times T$ such that for all $s \in S$ there is *exactly one* $t \in T$ such that $(s, t) \in f$.

We write $f(s)$ for the unique element related to s.

A **partial function** $f : S \nrightarrow T$ is a binary relation $f \subseteq S \times T$ such that for all $s \in S$ there is *at most one* $t \in T$ such that $(s, t) \in f$. That is, it is a function $f : S' \longrightarrow T$ for $S' \subseteq S$

Functions

$f : S \longrightarrow T$ describes pairing of the sets: it means that f assigns to every element $s \in S$ a unique element $t \in T$. To emphasise where a specific element is sent, we can write $f : x \mapsto y$, which means the same as $f(x) = y$

		Symbol	
S	**domain** of f	$\mathrm{Dom}(f)$	(inputs)
T	**co-domain** of f	$\mathrm{Codom}(f)$	(*possible* outputs)
$f(S)$	**image** of f	$\mathrm{Im}(f)$	(*actual* outputs)
$= \{ f(x) : x \in \mathrm{Dom}(f) \}$			

Important!

The domain and co-domain are critical aspects of a function's definition.

$$f : \mathbb{N} \to \mathbb{Z} \quad \text{given by} \quad f(x) \mapsto x^2$$

and

$$g : \mathbb{N} \to \mathbb{N} \quad \text{given by} \quad g(x) \mapsto x^2$$

are different functions even though they have the same behaviour!

Composition of Functions

Composition of functions is described as

$$g \circ f : x \mapsto g(f(x)), \quad \text{requiring } \operatorname{Im}(f) \subseteq \operatorname{Dom}(g)$$

Composition is associative

$$h \circ (g \circ f) = (h \circ g) \circ f, \quad \text{can write } h \circ g \circ f$$

Composition of Functions

If a function maps a set into itself, i.e. when $\mathrm{Dom}(f) = \mathrm{Codom}(f)$ (and thus $\mathrm{Im}(f) \subseteq \mathrm{Dom}(f)$), the function can be composed with itself — **iterated**

$$f \circ f, f \circ f \circ f, \ldots, \quad \text{also written } f^2, f^3, \ldots$$

Identity function on S

$$\mathrm{Id}_S(x) = x, x \in S; \mathrm{Dom}(i) = \mathrm{Codom}(i) = \mathrm{Im}(i) = S$$

For $g : S \longrightarrow T \quad g \circ \mathrm{Id}_S = g,\ \mathrm{Id}_T \circ g = g$

Extension: Composition of Binary Relations

If $R_1 \subseteq S \times T$ and $R_2 \subseteq T \times U$ then the composition of R_1 and R_2 is the relation:

$$R_1 ; R_2 := \{(a, c) : \text{ there is a } b \in T \text{ such that } (a, b) \in R_1 \text{ and } (b, c) \in R_2\}.$$

Note that if $f : S \to T$ and $g : T \to S$ are functions then $f ; g = g \circ f$.

Properties of Functions

Function is called **surjective** or **onto** if every element of the codomain is mapped to by at least one x in the domain, i.e.

$$\text{Im}(f) = \text{Codom}(f)$$

Examples (of functions that are surjective)

- $f : \mathbb{N} \longrightarrow \mathbb{N}$ with $f(x) \mapsto x$
- Floor, ceiling

Examples (of functions that are not surjective)

- $f : \mathbb{N} \longrightarrow \mathbb{N}$ with $f(x) \mapsto x^2$
- $f : \{a, \ldots, z\}^* \longrightarrow \{a, \ldots, z\}^*$ with $f(\omega) \mapsto a\omega e$

Injective Functions

Function is called **injective** or **1–1** (**one-to-one**) if different x implies different $f(x)$, i.e.

$$f(x) = f(y) \rightarrow x = y$$

Examples (of functions that are injective)

- $f : \mathbb{N} \longrightarrow \mathbb{N}$ with $f(x) \mapsto x$
- set complement (for a fixed universe)

Examples (of functions that are not injective)

- absolute value, floor, ceiling
- length of a word

Function is **bijective** if it is both surjective and injective.

Converse of a function

Question

$f^{\leftarrow}$ *is a relation; when is it a function?*

Question

$f^{\leftarrow}$ is a relation; when is it a function?

Answer

When f is a bijection.

Inverse Functions

Inverse function — $f^{-1} : T \longrightarrow S$;
for a given $f : S \longrightarrow T$ exists exactly
when f is bijective.

Image of a subdomain A under a function

$$f(A) = \{\, f(s) : s \in A \,\} = \{\, t \in T : t = f(s) \text{ for some } s \in A \,\}$$

Inverse image — $f^{\leftarrow}(B) = \{\, s \in S : f(s) \in B \,\} \subseteq S$;
it is defined for every f (recall: converse of a relation)

If f^{-1} exists then $f^{\leftarrow}(B) = f^{-1}(B)$

$f(\emptyset) = \emptyset, f^{\leftarrow}(\emptyset) = \emptyset$

Summary of topics

- Sets
- Formal languages
- Relations
- Functions
- Propositional Logic

Propositions

A sentence of a natural language (like English, Cantonese, Warlpiri) is *declarative*, or a *proposition*, if it can be meaningfully be said to be either true or false.

Examples

- Richard Nixon was president of Ecuador.
- A square root of 16 is 4.
- Euclid's program gets stuck in an infinite loop if you input 0.
- Whatever list of numbers you give as input to this program, it outputs the same list but in increasing order.
- $x^n + y^n = z^n$ has no nontrivial integer solutions for $n > 2$.
- 3 divides 24.
- K_5 is planar.

The following are *not* declarative sentences:

- Gubble gimble goo
- For Pete's sake, take out the garbage!
- Did you watch MediaWatch last week?
- Please waive the prerequisites for this subject for me.
- x divides y. — $R(x, y)$
- $x = 3$ and x divides 24. — $P(x)$

The following are *not* declarative sentences:

- Gubble gimble goo
- For Pete's sake, take out the garbage!
- Did you watch MediaWatch last week?
- Please waive the prerequisites for this subject for me.
- x divides y. — $R(x, y)$
- $x = 3$ and x divides 24. — $P(x)$

Declarative sentences in natural languages can be *compound* sentences, built out of other sentences.

Propositional Logic is a formal representation of some constructions for which the truth value of the compound sentence can be determined from the truth value of its components.

- Chef is a bit of a Romeo *and* Kenny is always getting killed.
- Either Bill is a liar *or* Hillary is innocent of Whitewater.
- *It is not the case that* this program always halts.

Not all constructions of natural language are truth-functional:

- *Obama believes that* Iran is developing nukes.
- *Chef said* they killed Kenny.
- This program always halts *because* it contains no loops.
- The disk crashed *after* I saved my file.

NB

Various **modal logics** *extend classical propositional logic to represent, and reason about, these and other constructions.*

The Three Basic Connectives of Propositional Logic

symbol	**text**
$\wedge$	"and", "but", ";", ":"
$\vee$	"or", "either ... or ..."
$\neg$	"not", "it is not the case that"

Truth tables:

A	B	A $\wedge$ B
F	F	F
F	T	F
T	F	F
T	T	T

A	B	A $\vee$ B
F	F	F
F	T	T
T	F	T
T	T	T

A	$\neg$ A
F	T
T	F

Applications I: Program Logic

Example

```
if x > 0 or (x <= 0 and y > 100):
```

Let $p \stackrel{\text{def}}{=} (\texttt{x > 0})$ and $q \stackrel{\text{def}}{=} (\texttt{y > 100})$

$p \lor (\neg p \land q)$

p	q	$\neg p$	$\neg p \land q$	$p \lor (\neg p \land q)$
F	F	T	F	F
F	T	T	T	T
T	F	F	F	T
T	T	F	F	T

This is equivalent to $p \lor q$. Hence the code can be simplified to

```
if x > 0 or y > 100:
```

Somewhat more controversially, consider the following constructions:

- if A then B
- A only if B
- B if A
- A implies B
- it follows from A that B
- whenever A, B
- A is a sufficient condition for B
- B is a necessary condition for A

Each has the property that if the whole statement is true, and A is true, then B is true.

Vacuous truth

How to interpret $A \to B$ when A is false?

$$A \to B \qquad \text{If } A \text{ (premise) then } B \text{ (conclusion)}$$

Material implication is false *only when* the premise holds and the conclusion does not.

If the premise is false, the implication is true no matter how absurd the conclusion is.

Both the following statements are true:

- If February has 30 days then March has 31 days.
- If February has 30 days then March has 42 days.

We can *approximate* the English meaning of $A \rightarrow B$ by "not (A and not B)" which has the following truth table:

A	B	A → B
F	F	T
F	T	T
T	F	F
T	T	T

While only an approximation to the English, 100+ years of experience have shown this to be adequate for capturing *mathematical reasoning*.
(Moral: mathematical reasoning does not need all the features of English.)

Examples

$$
\begin{aligned}
p &= \text{“you get an HD on your final exam”} \\
q &= \text{“you do every exercise in the book”} \\
r &= \text{“you get an HD in the course”}
\end{aligned}
$$

Translate into logical notation:

(a) You get an HD in the course although you do not do every exercise in the book.

(c) To get an HD in the course, you must get an HD on the exam.

(d) You get an HD on your exam, but you don't do every exercise in this book; nevertheless, you get an HD in this course.

Examples

$$\begin{aligned} p &= \text{"you get an HD on your final exam"} \\ q &= \text{"you do every exercise in the book"} \\ r &= \text{"you get an HD in the course"} \end{aligned}$$

Translate into logical notation:

(a) You get an HD in the course although you do not do every exercise in the book. $r \wedge \neg q$

(c) To get an HD in the course, you must get an HD on the exam. $r \to p$

(d) You get an HD on your exam, but you don't do every exercise in this book; nevertheless, you get an HD in this course. $p \wedge \neg q \wedge r$

Examples

$$\begin{aligned} p &= \text{"you get an HD on your final exam"} \\ q &= \text{"you do every exercise in the book"} \\ r &= \text{"you get an HD in the course"} \end{aligned}$$

Translate into logical notation:

(a) You get an HD in the course although you do not do every exercise in the book. $r \wedge \neg q$

(c) To get an HD in the course, you must get an HD on the exam.

$r \rightarrow p$

(d) You get an HD on your exam, but you don't do every exercise in this book; nevertheless, you get an HD in this course. $p \wedge \neg q \wedge r$

Examples

$$
\begin{aligned}
p &= \text{“you get an HD on your final exam”}\\
q &= \text{“you do every exercise in the book”}\\
r &= \text{“you get an HD in the course”}
\end{aligned}
$$

Translate into logical notation:

(a) You get an HD in the course although you do not do every exercise in the book. $r \wedge \neg q$

(c) To get an HD in the course, you must get an HD on the exam. $r \rightarrow p$

(d) You get an HD on your exam, but you don't do every exercise in this book; nevertheless, you get an HD in this course. $p \wedge \neg q \wedge r$

Examples

$$
\begin{aligned}
p &= \text{“you get an HD on your final exam”} \\
q &= \text{“you do every exercise in the book”} \\
r &= \text{“you get an HD in the course”}
\end{aligned}
$$

Translate into logical notation:

(a) You get an HD in the course although you do not do every exercise in the book. $r \wedge \neg q$

(c) To get an HD in the course, you must get an HD on the exam. $r \rightarrow p$

(d) You get an HD on your exam, but you don't do every exercise in this book; nevertheless, you get an HD in this course. $p \wedge \neg q \wedge r$

Unless

A unless B can be approximated as $\neg B \to A$

E.g.
I go swimming unless it rains = If it is not raining then I go swimming.
Correctness of the translation is perhaps easier to see in:
I don't go swimming unless the sun shines = If the sun does not shine then I don't go swimming.

Note that "I go swimming unless it rains, but sometimes I swim even though it is raining" makes sense, so the translation of "A unless B" should not imply $B \to \neg A$.

Just in case/if and only if

A just in case B usually means *A if, and only if, B*; written $A \leftrightarrow B$

The program terminates just in case the input is a positive number.
= The program terminates if, and only if, the input is positive.

I will have an entree just in case I won't have desert.
= If I have desert I will not have an entree and vice versa.

It has the following truth table:

A	B	A $\leftrightarrow$ B
F	F	T
F	T	F
T	F	F
T	T	T

Same as $(A \rightarrow B) \wedge (B \rightarrow A)$

A **Propositional formula** is made up of **propositional variables** and **logical connectives** $(\wedge, \vee, \neg, \rightarrow, \leftrightarrow)$.

A **truth assignment** assigns T or F to each propositional variable, and, using the logical connectives, gives a truth value to all propsitional formulas.

Logical Equivalence

Two formulas ϕ, ψ are **logically equivalent**, denoted $\phi \equiv \psi$ if they have the same truth value for all truth valuations.

Application: If ϕ and ψ are two formulae such that $\phi \equiv \psi$, then the digital circuits corresponding to ϕ and ψ compute the same function. Thus, proving equivalence of formulas can be used to *optimise* circuits.

Some Well-Known Equivalences

Excluded Middle	$p \vee \neg p \equiv \top$
Contradiction	$p \wedge \neg p \equiv \bot$
Identity	$p \vee \bot \equiv p$
	$p \wedge \top \equiv p$
	$p \vee \top \equiv \top$
	$p \wedge \bot \equiv \bot$
Idempotence	$p \vee p \equiv p$
	$p \wedge p \equiv p$
Double Negation	$\neg\neg p \equiv p$
Commutativity	$p \vee q \equiv q \vee p$
	$p \wedge q \equiv q \wedge p$

Associativity	$(p \vee q) \vee r \equiv p \vee (q \vee r)$
	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Distribution	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
De Morgan's laws	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
	$\neg(p \vee q) \equiv \neg p \wedge \neg q$
Implication	$p \to q \equiv \neg p \vee q$
	$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$

Satisfiability of Formulas

A formula is **satisfiable**, if it evaluates to T for *some* assignment of truth values to its basic propositions.

Example

A	B	$\neg(A \rightarrow B)$
F	F	F
F	T	F
T	F	T
T	T	F

Applications II: Constraint Satisfaction Problems

These are problems such as timetabling, activity planning, etc.
Many can be understood as showing that a formula is satisfiable.

Example

You are planning a party, but your friends are a bit touchy about who will be there.

1. If John comes, he will get very hostile if Sarah is there.
2. Sarah will only come if Kim will be there also.
3. Kim says she will not come unless John does.

Who can you invite without making someone unhappy?

Translation to logic: let J, S, K represent "John (Sarah, Kim) comes to the party". Then the constraints are:

1. $J \to \neg S$
2. $S \to K$
3. $K \to J$

Thus, for a successful party to be possible, we want the formula $\phi = (J \to \neg S) \wedge (S \to K) \wedge (K \to J)$ to be satisfiable.
Truth values for J, S, K making this true are called *satisfying assignments*, or *models*.

We figure out where the conjuncts are false, below. (so blank = T)

J	K	S	$J \to \neg S$	$S \to K$	$K \to J$	ϕ
F	F	F				
F	F	T		F		F
F	T	F			F	F
F	T	T			F	F
T	F	F				
T	F	T	F	F		F
T	T	F				
T	T	T	F			F

Conclusion: a party satisfying the constraints can be held. Invite nobody, or invite John only, or invite Kim and John.

Exercise

2.7.14 (supp)

Which of the following formulae are *always* true?

(a) $(p \wedge (p \rightarrow q)) \rightarrow q$ — always true

(b) $((p \vee q) \wedge \neg p) \rightarrow \neg q$ — not always true

(e) $((p \rightarrow q) \vee (q \rightarrow r)) \rightarrow (p \rightarrow r)$ — not always true

(f) $(p \wedge q) \rightarrow q$ — always true

Exercise

$\boxed{\text{2.7.14 (supp)}}$

Which of the following formulae are *always* true?

(a) $(p \wedge (p \to q)) \to q$ — always true

(b) $((p \vee q) \wedge \neg p) \to \neg q$ — not always true

(e) $((p \to q) \vee (q \to r)) \to (p \to r)$ — not always true

(f) $(p \wedge q) \to q$ — always true

Validity, Entailment, Arguments

An *argument* consists of a set of declarative sentences called *premises* and a declarative sentence called the *conclusion*.

Example

Premises:	Frank took the Ford or the Toyota.
	If Frank took the Ford he will be late.
	Frank is not late.
Conclusion:	Frank took the Toyota

An argument is *valid* if the conclusions are true *whenever* all the premises are true. Thus: if we believe the premises, we should also believe the conclusion.
(Note: we don't care what happens when one of the premises is false.)
Other ways of saying the same thing:

- The conclusion *logically follows* from the premises.
- The conclusion is a *logical consequence* of the premises.
- The premises **entail** the conclusion.

The argument above is valid. The following is invalid:

Example

Premises:	Frank took the Ford or the Toyota.
	If Frank took the Ford he will be late.
	Frank is late.
Conclusion:	Frank took the Ford.

For arguments in propositional logic, we can capture validity as follows:
Let $\phi_1, \ldots, \phi_n$ and ϕ be formulae of propositional logic. Draw a truth table with columns for each of $\phi_1, \ldots, \phi_n$ and ϕ.
The argument with premises $\phi_1, \ldots, \phi_n$ and conclusion ϕ is valid, denoted

$$\phi_1, \ldots, \phi_n \models \phi$$

if in every row of the truth table where $\phi_1, \ldots, \phi_n$ are all true, ϕ is true also.

We mark only true locations (blank = F)

Frd	$Tyta$	$Late$	$Frd \vee Tyta$	$Frd \rightarrow Late$	$\neg Late$	$Tyta$
F	F	F		T	T	
F	F	T		T		
F	T	F	T	T	T	T
F	T	T	T	T		T
T	F	F	T		T	
T	F	T	T	T		
T	T	F	T		T	T
T	T	T	T	T		T

This shows $Frd \vee Tyta, Frd \rightarrow Late, \neg Late \models Tyta$

The following row shows $Frd \lor Tyta$, $Frd \to Late$, $Late \not\models Frd$

Frd	$Tyta$	$Late$	$Frd \lor Tyta$	$Frd \to Late$	$Late$	Frd
F	T	T	T	T	T	F

Applications III: Reasoning About Requirements/Specifications

Suppose a set of English language requirements R for a software/hardware system can be formalised by a set of formulae $\{\phi_1, \dots \phi_n\}$.
Suppose C is a statement formalised by a formula ψ. Then

1. The requirements cannot be implemented if $\phi_1 \wedge \dots \wedge \phi_n$ is not satisfiable.
2. If $\phi_1, \dots \phi_n \models \psi$ then every correct implementation of the requirements R will be such that C is always true in the resulting system.
3. If $\phi_1, \dots \phi_{n-1} \models \phi_n$, then the condition ϕ_n of the specification is redundant and need not be stated in the specification.

Example

Requirements R: A burglar alarm system for a house is to operate as follows. The alarm should not sound unless the system has been armed or there is a fire. If the system has been armed and a door is disturbed, the alarm should ring. Irrespective of whether the system has been armed, the alarm should go off when there is a fire.

Conclusion C: If the alarm is ringing and there is no fire, then the system must have been armed.

Questions

1. Will every system correctly implementing requirements R satisfy C?
2. Is the final sentence of the requirements redundant?

Expressing the requirements as formulas of propositional logic, with

- S = the alarm sounds = the alarm rings
- A = the system is armed
- D = a door is disturbed
- F = there is a fire

we get

Requirements:

1. $S \to (A \vee F)$
2. $(A \wedge D) \to S$
3. $F \to S$

Conclusion: $(S \wedge \neg F) \to A$

Our two questions then correspond to

1. Does $S \to (A \vee F), (A \wedge D) \to S, F \to S \models (S \wedge \neg F) \to A$?
2. Does $S \to (A \vee F), (A \wedge D) \to S \models F \to S$?

Validity of Formulas

A formula ϕ is **valid**, or a **tautology**, denoted $\models \phi$, if it evaluates to T for *all* assignments of truth values to its basic propositions.

Example

A	B	$(A \to B) \to (\neg B \to \neg A)$
F	F	T
F	T	T
T	F	T
T	T	T

Validity, Equivalence and Entailment

Theorem

The following are equivalent:

- $\phi_1, \ldots \phi_n \models \psi$
- $\models (\phi_1 \wedge \ldots \wedge \phi_n) \rightarrow \psi$
- $\models \phi_1 \rightarrow (\phi_2 \rightarrow \ldots (\phi_n \rightarrow \psi) \ldots)$

Theorem

$\phi \equiv \psi$ *if and only if* $\models \phi \leftrightarrow \psi$

Deeper reasoning

Entailment captures a form of logical reasoning, but it cannot handle relatively simple logical arguments like the following:

1. Socrates is a man
2. All men is mortal
3. Therefore Socrates is mortal

We need to add **expressiveness** to propositional logic so that we can capture notions such as the *relation* between man and men in the first two statements; and the *quantified* statement "all men".

NB

Adding expressiveness comes at a cost: it is now more difficult to determine truth values.

Predicates

Predicates are functions that take inputs from a set of individuals and return either true or false – i.e. they are relations between the individuals. Predicates enable us to establish relationships between different propositions, such as the man/men connection between the first and second propositions on the previous slide, allowing deeper reasoning than propositional logic can give.

Quantifiers

Quantifiers allow us to make quantified statements over predicates, e.g.

> *"If there exists a satisfying assignment ..."*

or

> *"Every natural number greater than 2 ..."*

The two standard quantifiers are

- $\forall$: "for all", "for any", "every"
- $\exists$: "there exists", "there is", "for some", "at least one"

Example

Goldbach's conjecture

$$\forall n \in 2\mathbb{N}\,(n > 2 \to \exists p, q \in \mathbb{N}\,(p, q \in \textsc{Primes} \land n = p + q))$$

Predicate logic

Predicate (or first-order) logic extends propositional logic by adding predicates and quantifiers.
Propositional logic is about reasoning with propositions: statements that are either true or false. Predicate logic extends propositional logic by examining *why* propositions might be true or false.

Summary of topics

- Sets
- Formal languages
- Relations
- Functions
- Propositional (and Predicate) Logic

What is assessible?

Everything. However...

Summary of topics

- Sets
- Formal languages
- Relations
- Functions
- Propositional (and Predicate) Logic

What is assessible?

Everything. However...

Summary of topics

- Sets
- Formal languages
- Relations
- Functions
- Propositional (and Predicate) Logic

What is assessible?

Everything. However...