# Sets

A **set** is a collection of (mathematical) objects.

Each object $x$ is either contained in a set $S$ or not.

We write $x \in S$ to mean '$x$ is an element of $S$' or '$x$ is in $S$' or '$S$ contains $x$'

We write $x \notin S$ to mean '$x$ is not an element of $S$' or '$x$ is not in $S$' or '$S$ does not contain $x$'

**Finite sets:**

- $\emptyset$ the **empty set**. It contains no objects.
  - $*$ In particular $\emptyset \notin \emptyset$
- $\{x\}$ the set containing only $x$
- $\{x, y, z\}$ the set containing $x$, $y$, and $z$, but nothing else.

**Some infinite sets:**

- $\mathbb{N}$ contains the natural numbers: $0$, $1$, $2$, $3$, etc. **Note** $0$ **is included!**
- $\mathbb{Z}$ contains the integers: $0$, $-1$, $1$, $-2$, $2$, $-3$, $3$, etc.
- $\mathbb{R}$ contains all the real numbers.
- Don't confuse zero with the empty set: $0 \neq \emptyset$.

**Equality:** Two sets are considered equal ($S = T$) iff they contain exactly the same objects.

- Therefore there is only one empty set (all empty sets are equal).

**Union**: $S \cup T$ is the set that contains all objects contained either in $S$ or in $T$ or in both.

- $x \in (S \cup T)$ exactly if $\boxed{\qquad\qquad\qquad\qquad}$

**Intersection**: $S \cap T$ is the set that contains all objects contained both in $S$ and in $T$.

- $x \in (S \cap T)$ exactly if $\boxed{\qquad\qquad\qquad\qquad}$

**Subtraction**: $S - T$ is the set that contains all objects in $S$ that are not in $T$.

- $x \in (S - T)$ exactly if $\boxed{\qquad\qquad\qquad\qquad}$

**Subsets**:

- $S \subseteq T$ means '$S$ is a **subset** of $T$', i.e., every object in $S$ is also in $T$.
- In particular, $S \subseteq S$ and $\emptyset \subseteq S$, for any set $S$.
- E.g. $\mathbb{N} \subseteq \mathbb{Z}$

Don't confuse singleton sets with their single element.

- Is $1 \in \{1, 2, 3\}$ ? $\boxed{\qquad}$
- Is $\{1\} \in \{1, 2, 3\}$ ? $\boxed{\qquad}$
- Is $\{1\} \in \{\{1\}, \{2\}, \{3\}\}$ ? $\boxed{\qquad}$
- Is $1 \in \{\{1\}, \{2\}, \{3\}\}$ ? $\boxed{\qquad}$
- Is $\emptyset \subseteq \{1, 2, 3\}$ ? $\boxed{\qquad}$
- Is $\{\emptyset\} \subseteq \{1, 2, 3\}$ ? $\boxed{\qquad}$

**Python Note:** Python has two types of objects for representing sets: "`set`" and "`frozenset`". Objects of type `set` are mutable. Objects of type `frozenset` are immutable.

## Set Comprehension (Set Builder Notation)

**Filtering**. Let $\mathcal{A}$ be some description of variable $x$ (i.e. a Boolean expression; e.g. $x > 0$.)

- $\{x \in S \mid \mathcal{A}\}$ means 'that subset of $S$ containing all and only elements described by $\mathcal{A}$'.
- For example:
  - $\{x \in \mathbb{R} \mid x > 0\}$ is the set of positive real numbers
  - $\{x \in \mathbb{N} \mid x/3 \in \mathbb{N}\}$ is the set of natural number that are multiples of $3$.

**Mapping**. Let $\mathcal{F}$ be some expresion involving $x$.

- $\{\mathcal{F} \mid x \in S\}$ means the set of all values of $\mathcal{F}$ where $x$ is some element of $S$.
- For example:
  - $\{2x \mid x \in \mathbb{N}\}$ is the set of even natural numbers
  - $\{f(x) \mid x \in \mathbb{Z}\}$ is the set of values of $f$ when its argument is an integer.

**Python Note:** Python supports both filtering and mapping. For example

- Filtering: `{ x for x in R if x > 0 }`
- Mapping: `{ 2*x for x in S }`
- Both: `{ 2*x for x in S if isOdd(x) }`

## Sets of integers

- $\{0, ..n\}$ the first $n$ natural numbers
  - $\{0, ..2^8\}$ integers representable by $8$ bits unsigned.
  - $\{0, ..\operatorname{length}(s)\}$ the indecies of sequence $s$.
- $\{i, ..j\}$ all integers from $i$ up to but not including $j$.
  - $\{-2^7, ..2^7\}$ integers representable by 8 bits 2's-complement.
  - Use filtering to describe $\{i, ..j\}$
  - 
- $\{i, .., j\}$ all integers from $i$ up to and including $j$.
  - $\{-1, .., 1\} = \{0, -1, 1\}$
  - Use filtering to describe $\{i, .., j\}$
  -

## Pairs and tuples

**Pairs, triples, and tuples:**

- $(x, y)$ is a **pair** consisting of $x$ on the left and $y$ on the right.
- Note that $(x, y) \neq (y, x)$ unless $x = y$.
- We can also have **triple**s $(x, y, z)$ and in general $n$-**tuple**s for $n \geq 2$.
- E.g. $(1, \pi, \mathfrak{true}, \text{‘a’}, \emptyset)$ is a 5-tuple.

**Python note:** In ordinary math, there are no 1-tuples. I.e., $x = (x)$. However, in Python, there are 1-tuples, written somewhat perversely as `(x,)`. For example in Python `len( (42,) ) == 1`. What Python calls "tuples" are, in mathematical terms, "sequences". More below.

**Cartesian product**:

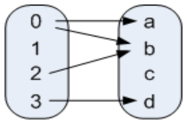
- $S \times T$ is the set of all pairs $(x, y)$ such that $x \in S$ and $y \in T$.
  - E.g.
    $$\{0, 1, 2\} \times \{\text{‘a’}, \text{‘b’}\}$$
    $$= \boxed{\phantom{xxxxxxxxxxxxxxxxxxxxxxxx}}$$
  - E.g. $\mathbb{R} \times \mathbb{R}$ is the Cartesian plane.
  - E.g. $\left\{(x, y) \in \mathbb{R} \times \mathbb{R} \mid x^2 + y^2 \leq 1\right\}$ is the unit disk at the origin.
- $S \times T \times U$ is the set of all triples $(x, y, z)$ such that $x \in S$, $y \in T$, and $x \in U$.
- Etc.

# Relations and functions.

- A **binary relation** is any triple $(S, T, G)$ where $S$ and $T$ are sets and $G \subseteq S \times T$.
- We call $S$ the **source**, $T$ the **target**, and $G$ the **graph** of the binary relation.
  - ∗ Example: Let $S = \{0, 1, 2, 3\}$ , $T = \{$‘a’, ‘b’, ‘c’, ‘d’$\}$,

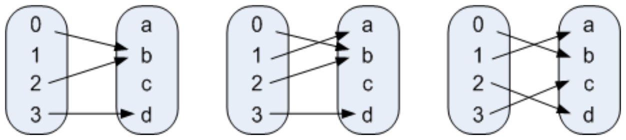
    $G = \{(0,$ ‘a’$), (0,$ ‘b’$), (2,$ ‘b’$), (3,$ ‘d’$)\}$

    Then $(S, T, G)$ is a binary relation, illustrated as

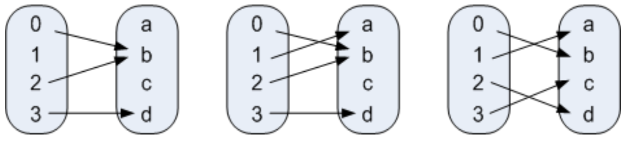
    

    Binary Relation

    Here are some more (illustrations of) binary relations.

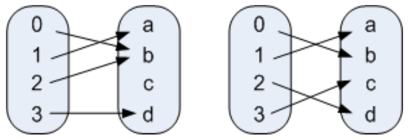
    

- A **partial function** $(S, T, G)$ is a binary relation such that, for each $x \in S$, there is at most one $y$ such that $(x, y) \in G$.
  - ∗ Example: Here are some examples of partial functions

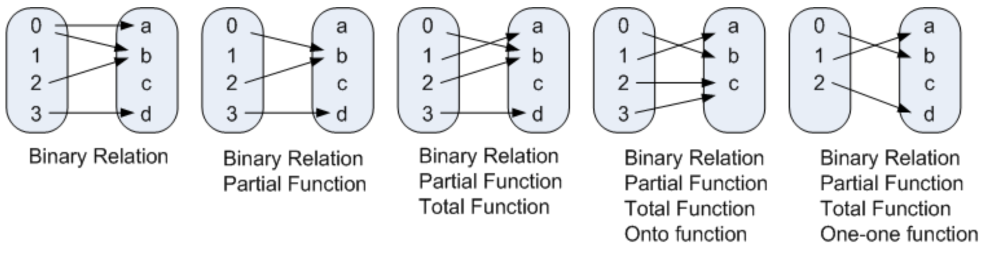
    

- A **total function** $(S, T, G)$ is a relation such that, for each $x \in S$, there is exactly one $y$ such that $(x, y) \in G$.
- (Note that each ‘total function’ is also a ‘partial function’!)

∗ Examples:



• Examples of binary relations:



• Let $f = (S, T, G)$ be a binary relation and $x \in S$.
  ∗ We say that $f$ is **defined for** $x$ iff there is a $y \in T$ such that $(x, y) \in G$.

• Suppose $f = (S, T, G)$ is a partial function and $f$ is defined for $x$:
  ∗ We say $f$ **applied to** $x$ to mean 'that $y \in T$ such that $(x, y) \in G$'.
  ∗ We write $f(x)$
  ∗ Of course $f(x)$ only makes sense if $f$ is defined for $x$.
  ∗ If $f$ is a total function, then $f(x)$ is defined for all $x$ in $T$.

• $S \to T$ is the set of all total functions with source $S$ and target $T$.

• $S \overset{\text{par}}{\to} T$ is the set of all partial functions with source $S$ and target $T$.

• Let $g = (S, T, G)$ and $f = (T, U, H)$ be partial functions:
  ∗ $f \circ g$ is the **composition** of $f$ and $g$ and is a relation

with source $S$ and target $U$ with graph

$$\{(s,u) \mid \exists t \in T \cdot (s,t) \in G \wedge (t,u) \in H\}$$

- $*$ If $f$ and $g$ are both partial functions, $f \circ g$ is a partial function.
- $*$ If $g(x)$ is defined and $f(g(x))$ is defined, then $(f \circ g)$ is defined at $x$ and $f(g(x)) = (f \circ g)(x)$.
- $*$ So if $f$ and $g$ are both total functions, $f \circ g$ is a total function.
- $*$ Sometimes I write $fg$ instead of $f \circ g$.

## Domain and Range

The **domain** of relation $R$ is the set of elements that map to something

$$\mathrm{dom}(R) = \{x \in S \mid R \text{ is defined for } x\}$$

(For a total function $f : S \to T$ the domain is the same as the source $S$, i.e,. $\mathrm{dom}(f) = S$.)

The **range** of $R$ is the set of elements that appear as the right component of a pair in the graph

$$\mathrm{rng}(R) = \{y \in T \mid \text{there is an } x \in S \text{ such that } (x,y) \in G\}$$

(If the range of a relation is its target, we say the relation is ‘onto’.)

## More Examples

- Consider $(\mathbb{Z}, \mathbb{Z}, J)$ and

$$J = \{(a,b) \in \mathbb{Z} \times \mathbb{Z} \mid b = a \times a\}$$

  - $*$ This is a total function (and hence also a partial function and a relation).

  * Its domain is $\mathbb{Z}$
  * Its range is $\{0, 1, 4, 9, ...\}$
- Consider $(\mathbb{R}, \mathbb{R}, K)$ where
$$K = \{(x, y) \in \mathbb{R} \times \mathbb{R} \mid x \times y = 1\}$$
  * This is a partial function. It is not total since there is no $(x, y)$ pair with $x = 0$.
  * Its domain and range is $\mathbb{R} - \{0.0\}$
- Consider $(\mathbb{R}, \mathbb{R}, L)$ where
$$L = \{(x, y) \in \mathbb{R} \times \mathbb{R} \mid y \times y = x\}$$
- This is not a function since we have $(4, 2)$ and $(4, -2)$ in the graph.
  * It has a domain of $\{x \in \mathbb{R} \mid x \geq 0\}$ and a range of $\mathbb{R}$.

## A digression on terminology (not covered in class)

The words ‘**relation**’, **function**’ and ‘**partial function**’ are defined differently by different people.

Some people define ‘**function**’ to mean (what I’ve called) ‘**total function**’.

Some people define ‘**function**’ to mean (what I’ve called) ‘**partial function**’.

Some people use the words ‘**partial function**’ only for partial functions that are not total.

Furthermore some people aren’t particularly consistent. For example, both the Mathworld website and the Wikipedia entry on ‘function’ manage to contradict themselves. (At least at the time I am writing this.)

When it matters, I will try to avoid omitting the words ‘total’ or ‘partial’.

Also the words ‘**domain**’ and ‘**range**’ are used in many ways. E.g. what I call the source and target, others call the domain and the co-domain.

You should be aware that different authors will use these words somewhat differently.

## Sequences

A **finite sequence** is a total function whose source is $\{0, ..n\}$, for some $n \in \mathbb{N}$.

I'll write $[a, b, c]$ for a finite sequence. In particular $[\,]$ is the finite sequence of length $0$.

I'll write $\text{length}(s)$ or $s.\text{length}$ for the length of a finite sequence.

I'll write $s\hat{\ }t$ for the catenation of two sequences.

- For example $[a, b, c]\hat{\ }[d, e, f]$ is $[a, b, c, d, e, f]$.

A sequence of ascending integers $[2, ..5] = [2, 3, 4]$.

If we want to include both end points: $[2, .., 5] = [2, 3, 4, 5]$

The length of $[i, ..j]$ is $\max(0, j - i)$.

The length of $[i, .., j]$ is $\max(0, j + 1 - i)$.

Items.

- Indexing is from 0.
- If $s = [a, b, c, d, e, f]$ then $s(2) = c$
- $s$ is defined for $i$ iff $0 \le i < \text{length}(s)$

Subsequence.

- If $s = [a, b, c, d, e, f]$ then $s[1, 1, 3, 2] = [b, b, d, c]$.
- $s\,t$ is defined iff $\forall k \in \{0, ..\text{length}(t)\} \cdot 0 \le t(k) < \text{length}(s)$
- If defined, its length is $\text{length}(t)$ and $(s\,t)(k) = s(t(k))$,for all $k \in \{0, ..\text{length}(t)\}$

Segments are contiguous subsequences

- If $s = [a, b, c, d, e, f]$ then $s[2, ..5] = [c, d, e]$
- If $s = [a, b, c, d, e, f]$ then $s[2, .., 5] = [c, d, e, f]$

Item sets: Sometimes it's useful to have a set of items based on a sequence and a set of indeces

- If $s = [a, b, c, d, e, f]$ then $s\{2, ..5\} = \{c, d, e\}$
- In general, $s\,T$ is $\{s(i) \mid i \in T\}$

**Python note:** Python has two varieties of generic sequences: "`tuple`" and "`list`". Objects of type `tuple` are imutable. Objects of type `list` are mutable. In also has some other sequence types. E.g. "`string`" objects are immutable sequences of characters.

## Graphs (for reference, not covered in class)

### Directed graphs

A **directed graph** $G = (N, E, \overleftarrow{\ }, \overrightarrow{\ })$ consists of a set of **nodes** $N$, a set of **edges** $E$ and two total functions $\overleftarrow{\ }$ and $\overrightarrow{\ }$ from $E \to N$. For each edge, $\overleftarrow{e}$ is its **source** node, while $\overrightarrow{e}$ is its **target** node.

An edge $e$ such that $\overleftarrow{e} = \overrightarrow{e}$ is called a **loop**.

A directed graph is **simple** if for any given source and target nodes there is at most one edge. I.e., if for all $(u, v) \in N \times N$, there is at most one $e \in E$, such that $u = \overleftarrow{e}$ and $v = \overrightarrow{e}$.

When $E \subseteq N \times N$ and, for each $(u, v) \in E$, $\overleftarrow{(u, v)} = u$ and $\overrightarrow{(u, v)} = v$, the graph is simple and is usually written as $G = (N, E)$.

### Undirected graphs

For a set $N$, $\{\{u, v\} \mid u, v \in N\}$ is the set of subsets of $N$ of size 1 or 2.

An **undirected graph** $G = (N, E, \overleftrightarrow{\ })$ consists of a set of **nodes** $N$, a set of **edges** $E$ and an **incidence** function $\overleftrightarrow{\ } : E \to \{\{u, v\} \mid u, v \in N\}$. If $u \in \overleftrightarrow{e}$ then $u$ is an **endpoint** of $e$.

An edge such that $|\overleftrightarrow{e}| = 1$ is called a **loop**.

An undirected graph is **simple** if each set in $\{\{u,v\} \mid u,v \in N\}$ is $\overleftrightarrow{e}$ for at most one edge $e$. (I.e., if $\leftrightarrow$ is one-one.)

When $E \subseteq \{\{u,v\} \mid u,v \in N\}$ and $\overleftrightarrow{e} = e$, for each $e \in E$, the graph is simple and is usually written $G = (N, E)$.

## Paths

A **path** in a directed graph is an alternating sequence of nodes and edges, starting and ending with a node

$$[u_0, e_0, u_1, ..., e_{n-1}, u_n]$$

such that, for each $i$, $u_i = \overleftarrow{e_i}$ and $\overrightarrow{e_i} = u_{i+1}$.

When the graph is undirected, the requirement is that, for each $i$, $\overleftrightarrow{e_i} = \{u_i, u_{i+1}\}$.

The length of the path is the number of edges, $n$.

The path is said to be a **cycle** (or **circuit**) if $u_0 = u_n$.

Note that for any $u \in N$, $[u]$ is a path of length 0!

# Propositional logic

We assume a set $\mathbb{B}$ of size 2 $\mathbb{B} = \{\mathfrak{true}, \mathfrak{false}\}$

### Implication

Define a function $(\Rightarrow) \in \mathbb{B} \times \mathbb{B} \to \mathbb{B}$. so that, for all $q \in \mathbb{B}$, we have the following laws

$$(\mathfrak{false} \Rightarrow q) = \mathfrak{true} \text{ “False implies anything”}$$
$$(\mathfrak{true} \Rightarrow q) = q \text{ “Identity”}$$

In table form

| $p$ | $q$ | $p \Rightarrow q$ |
|---|---|---|
| $\mathfrak{false}$ | $\mathfrak{false}$ | |
| $\mathfrak{false}$ | $\mathfrak{true}$ | |
| $\mathfrak{true}$ | $\mathfrak{false}$ | |
| $\mathfrak{true}$ | $\mathfrak{true}$ | |

$\Rightarrow$ is called **implication**.

Its left operand is called the **antecedent** and its right operand is called the **consequent**.

Some laws

$$(p \Rightarrow p) = \mathfrak{true} \text{ Reflexivity}$$
$$(p \Rightarrow \mathfrak{true}) = \mathfrak{true} \text{ Domination}$$

### Negation

Define $(\neg) \in \mathbb{B} \to \mathbb{B}$ such that

$$\neg p = (p \Rightarrow \mathfrak{false}) \text{, for all } p \in \mathbb{B}$$

In table form

| $p$ | $\neg p$ |
|---|---|
| $\mathfrak{false}$ | $\mathfrak{true}$ |
| $\mathfrak{true}$ | $\mathfrak{false}$ |

Some laws

$$\neg\neg p = p \text{ Involution}$$
$$(p \Rightarrow q) = (\neg q \Rightarrow \neg p) \text{ Contrapositive}$$
$$(p \Rightarrow \mathfrak{false}) = \neg p \text{ Anti-identity}$$

## Conjunction (AND) and disjunction (OR)

Define **disjunction** (OR) by

$$(\vee) \in \mathbb{B} \times \mathbb{B} \to \mathbb{B}$$
$$(p \vee q) = \neg p \Rightarrow q$$

and **conjunction** (AND) by

$$(\wedge) \in \mathbb{B} \times \mathbb{B} \to \mathbb{B}$$
$$(p \wedge q) = \neg (p \Rightarrow \neg q)$$

Some laws

$$\left.\begin{array}{l}(\mathfrak{true} \wedge p) = p \\ (\mathfrak{false} \vee p) = p\end{array}\right\} \text{Identity}$$

$$\left.\begin{array}{l}(\mathfrak{false} \wedge p) = \mathfrak{false} \\ (\mathfrak{true} \vee p) = \mathfrak{true}\end{array}\right\} \text{Domination}$$

$$\left.\begin{array}{l}(p \wedge p) = p \\ (p \vee p) = p\end{array}\right\} \text{Idempotence}$$

$$\left.\begin{array}{l}(p \wedge q) = (q \wedge p) \\ (p \vee q) = (q \vee p)\end{array}\right\} \text{Commutativity}$$

$$\left.\begin{array}{l}(p \wedge q) \wedge r = p \wedge (q \wedge r) \\ (p \vee q) \vee r = p \vee (q \vee r)\end{array}\right\} \text{Associativity}$$

$$\left.\begin{array}{l}(p \wedge (q \vee r)) = ((p \wedge q) \vee (p \wedge r)) \\ (p \vee (q \wedge r)) = ((p \vee q) \wedge (p \vee r))\end{array}\right\} \text{Distributivity}$$

$$\left.\begin{array}{l}(p \wedge \neg p) = \mathfrak{false} \\ (p \vee \neg p) = \mathfrak{true}\end{array}\right\} \left\{\begin{array}{l}\text{law of contradiction} \\ \text{law of excluded middle}\end{array}\right.$$

$$\left.\begin{array}{l}\neg(p \wedge q) = \neg p \vee \neg q \\ \neg(p \vee q) = \neg p \wedge \neg q\end{array}\right\} \text{De Morgan's laws}$$

$(p \Rightarrow q) = (\neg p \vee q)$ Material implication

$(p \Rightarrow q) = (\neg p \Rightarrow \neg q)$ Contrapositive law

$(p \wedge q \Rightarrow r) = (p \Rightarrow (q \Rightarrow r))$ Shunting

$(p \wedge q \Rightarrow r) = ((p \Rightarrow r) \vee (q \Rightarrow r))$ Distributivity

$(p \vee q \Rightarrow r) = ((p \Rightarrow r) \wedge (q \Rightarrow r))$ Distributivity

$(p \Rightarrow q \wedge r) = ((p \Rightarrow q) \wedge (p \Rightarrow r))$ Distributivity

$(p \Rightarrow q \vee r) = ((p \Rightarrow q) \vee (p \Rightarrow r))$ Distributivity

if $p \Rightarrow q$ and $q \Rightarrow r$ then $p \Rightarrow r$ Transitivity

The operands of $\wedge$ are called **conjuncts**.

The operands of $\vee$ are called **disjuncts**.

## Equivalence and XOR

Define

$$\begin{array}{rcl}(p \Leftrightarrow q) & = & (p \Rightarrow q) \wedge (q \Rightarrow p) \\ (p \not\Leftrightarrow q) & = & \neg(p \Leftrightarrow q)\end{array}$$

$(p \Leftrightarrow q)$ is often called **equivalence.** It is really just **equality** for Boolean values.

$(p \not\Leftrightarrow q)$ is called **exclusive or**

## Notation

| Course | Digital Logic | C/C++/ Java | C/C++/Java bitwise | Other |
|---|---|---|---|---|
| $\Rightarrow$ | | | | $\supset, \rightarrow$ |
| $\wedge$ | $\cdot$ | && | & | |
| $\vee$ | $+$ | \|\| | \| | |
| $\Leftrightarrow$ | | == | | $\leftrightarrow, \equiv$ |
| $\not\Leftrightarrow$ | $\oplus$ | != | ^ | + |
| $\neg$ | $\overline{\phantom{x}}$ | ! | ~ | $\sim$ |

## Substitution

### Free and bound occurrences of variables

In Engineering, we often use variables to represent quantities in the real-world and boolean expressions containing variables to represent constraints on those quantities, imposed by nature or by an engineered system. For example, we might write

$$0 \le x < 1$$

to express that the $x$ coordinate of the position of something (say a robot's hand) is constrained within certain limits. A constraint

$$0 \le y < 1$$

means something quite different. So we can conclude that **the names matter**. We call such occurrences of a variable "**free**".

Now consider the following pairs of expressions

- $z = \sum_{i=0}^{N} f(i)$ and $z = \sum_{j=0}^{N} f(j)$
- $z < \int_0^\infty f(u)\,\mathrm{d}u$ and $z < \int_0^\infty f(v)\,\mathrm{d}v$
- $\{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \le 1.0\}$ and $\{(a, b) \in \mathbb{R}^2 \mid a^2 + b^2 \le 1.0\}$

In each case, the two parts of the pair express the same constraint: they are equivalent.

In these cases the variables $i$, $j$ , $u$, $v$, $x$, $y$, $a$, and $b$ are internal to the expression. They don't indicate anything outside of the expression.

Such occurrences of variables are called “**bound**”.

An analogous situation comes up in software.

The two subroutines

```
void f() { ++i ; }
```

and

```
void f() { ++j ; }
```

are *not* equivalent. The occurrences of `i` and `j` are free.

The two subroutines

```
int g(int i) { return i+1 ; }
```

and

```
int g(int j) { return j+1 ; }
```

are equivalent. The occurrences of `i` and `j` are bound.

## Single variable substitution

Suppose that $\mathcal{E}$ is an expression and that $\mathcal{V}$ is a variable.

We'll write $\mathcal{E}[\mathcal{V} : \mathcal{F}]$ for the expression obtained by replacing every free occurrence of variable $\mathcal{V}$ in $\mathcal{E}$ with expession $(\mathcal{F})$.

Examples

- $(x/y)[x : y + z]$ is $(y + z)/y$
- $(0 \le i < N \wedge A[i] = 0)[i : i + 1]$ is
$$0 \le i + 1 < N \wedge A[i + 1] = 0$$

## Multiple variables

We sometimes need to replace a number of variables at once.

We'll write $\mathcal{E}[\mathcal{V}_0, \mathcal{V}_1, \cdots, \mathcal{V}_{n-1} : \mathcal{F}_0, \mathcal{F}_1, \cdots, \mathcal{F}_{n-1}]$ to mean the simultaneous replacement of $n$ distinct variables by $n$ expressions.

Example

- $(x/y)[x, y : y, x]$ is $(y/x)$
- whereas $((x/y)[x : y])[y : x]$ is $(x/x)$

**Substitution and bound variables**

Making the same substitution in two equivalent expressions must give two equivalent expressions.

Thus we have to be a bit careful about exactly how substitution is defined.

In making substitutions we do not substitute for bound variables. For example in the expression

$$\sum_{i=0}^{N-1} f(i)$$

the variable $i$ is bound, so we don't substitute for it. Thus

$$\left(\sum_{i=0}^{N-1} f(i)\right)[f, i : g, j+1] \text{ is } \sum_{i=0}^{N-1} g(i)$$

Furthermore, it may be necessary to rename bound variables in order to avoid variables free in $\mathcal{F}$ from being "captured". For example

$$\left(\sum_{i=0}^{N-1}(k \times i)\right)[k : i+1] \text{ is } \left(\sum_{j=0}^{N-1}(k \times j)\right)[k : i+1]$$

$$\text{which is} \sum_{j=0}^{N-1}((i+1) \times j)$$

Note that I had to rename $i$ to $j$ to avoid conflict with the $i$ in the replacement expression.

### Notations

Different authors use different notations for substitution.

- Some writers write $\mathcal{E}(\mathcal{V}/\mathcal{F})$ while others write $\mathcal{E}(\mathcal{F}/\mathcal{V})$
- The most common notation is $\mathcal{E}^{\mathcal{V}}_{\mathcal{F}}$ .
- I use $\mathcal{E}[\mathcal{V} : \mathcal{F}]$ because it is hard to mistake for anything else.

### One-point laws

The substitution notation lets us express some useful laws called "one-point laws".

Consider $(\mathcal{V} = \mathcal{F}) \Rightarrow \mathcal{E}$ where $\mathcal{V}$ is a variable and $\mathcal{E}$ and $\mathcal{F}$ are expressions. If $\mathcal{V} \neq \mathcal{F}$ then the value of $\mathcal{E}$ doesn't matter, the implication will be $\mathfrak{true}$ regardless of the value of $\mathcal{E}$.

In the case of $\mathcal{V} = \mathcal{F}$, we need only worry about the value of $\mathcal{E}$ under the assumption that $\mathcal{V} = \mathcal{F}$.

The same reasoning applies to an expression $(\mathcal{V} = \mathcal{F}) \wedge \mathcal{E}$.

The one point laws can be expressed as:

$$((\mathcal{V} = \mathcal{F}) \Rightarrow \mathcal{E}) = ((\mathcal{V} = \mathcal{F}) \Rightarrow \mathcal{E}[\mathcal{V} : \mathcal{F}])$$

and

$$((\mathcal{V} = \mathcal{F}) \wedge \mathcal{E}) = ((\mathcal{V} = \mathcal{F}) \wedge \mathcal{E}[\mathcal{V} : \mathcal{F}])$$

## Substitutions quiz

Simplify as much as you can:

- $(0 \leq x < 10)\,[x : -x]$
- $i = 0 \Rightarrow a(i) > i$
- For these three assume $a$ is a sequence of length $n$ and $j < n$
  * $\left(s = \sum_{i=0}^{j} a(i)\right)[s : s + a(j)][j : j + 1]$
  * $\left(s = \sum_{i=0}^{j} a(i)\right)[j : j + 1][s : s + a(j)]$
  * $\left(s = \sum_{i=0}^{j} a(i)\right)[s, j : s + a(j), j + 1]$
- For this one, assume $a$ is a sequence of length $n$,

$$\left(j < n \wedge s = \sum_{i=0}^{j-1} a(i)\right) \Rightarrow \left(s = \sum_{i=0}^{j-1} a(i)\right)[s, j : s{+}a(j), j{+}1]$$

# Predicate Logic

In natural language, one often wants to express declarations such as

- All flavours of ice-cream are good.
- Some people like peanut butter.
- The Q output is always equal to the D input of the previous cycle.
- The system will be in the initial state within 5 seconds of the reset button being depressed.

To treat such sentences mathematically we extend logic with "**quantifiers**"

- $\forall$, pronounced "for all", and
- $\exists$, pronounced "exists".

You can say that $\forall$ and $\exists$ have the same relationship to $\wedge$ and $\vee$ (respectively) as $\sum$ has to $+$.

We will extend our 2-valued propositional logic to deal with the quantifiers.

## The Quantifiers $\forall$ and $\exists$

Suppose that $S$ is a finite set, for example $\{0, 1, 2, 3\}$, and $\mathcal{A}$ is a boolean expression then we write

$$\forall \mathcal{V} \in S \cdot \mathcal{A}$$

mean

$$\mathcal{A}[\mathcal{V} : 0] \wedge \mathcal{A}[\mathcal{V} : 1] \wedge \mathcal{A}[\mathcal{V} : 2] \wedge \mathcal{A}[\mathcal{V} : 3]$$

and we write

$$\exists \mathcal{V} \in S \cdot \mathcal{B}$$

to mean

$$\mathcal{B}[\mathcal{V} : 0] \vee \mathcal{B}[\mathcal{V} : 1] \vee \mathcal{B}[\mathcal{V} : 2] \vee \mathcal{B}[\mathcal{V} : 3]$$

just as we would write

$$\sum_{\mathcal{V}=0}^{3} \mathcal{E}$$

to mean

$$\mathcal{E}[\mathcal{V} : 0] + \mathcal{E}[\mathcal{V} : 1] + \mathcal{E}[\mathcal{V} : 2] + \mathcal{E}[\mathcal{V} : 3]$$

where $\mathcal{E}$ is some numerical expression.

The quantifier $\forall$ is pronounced “for all”.

The quantifier $\exists$ is pronounced “there exists a”.

As long as the set $S$ is finite, $\forall$ and $\exists$ are convenient notations, but not very interesting, as they don’t allow us to do any thing new.

But, if we allow $S$ to be an infinite set, then we have something very interesting.

For example consider the set $\mathbb{N} = \{0, 1, 2, ...\}$ then

$$(\forall \mathcal{V} \in \mathbb{N} \cdot \mathcal{A}) = \mathcal{A}[\mathcal{V} : 0] \wedge \mathcal{A}[\mathcal{V} : 1] \wedge \mathcal{A}[\mathcal{V} : 2] \wedge \cdots$$

and

$$(\exists \mathcal{V} \in \mathbb{N} \cdot \mathcal{A}) = \mathcal{A}[\mathcal{V} : 0] \vee \mathcal{A}[\mathcal{V} : 1] \vee \mathcal{A}[\mathcal{V} : 2] \vee \cdots$$

In general

- $\forall \mathcal{V} \in S \cdot \mathcal{A}$ is $\mathfrak{false}$ if $\mathcal{A}[\mathcal{V} : y]$ is $\mathfrak{false}$ for at least one value $y \in S$, otherwise it is $\mathfrak{true}$.
- $\exists \mathcal{V} \in S \cdot \mathcal{A}$ is $\mathfrak{true}$ if $\mathcal{A}[x : y]$ is $\mathfrak{true}$ for at least one value $y \in S$, otherwise it is $\mathfrak{false}$.

**Some examples:**

Suppose

- $F$ — a set of flavours.
- $iceCream(f)$ — the variety of ice-cream having flavour $f$.
- $P$ — the set of all people.
- $peanutButter$ — peanut butter
- $like(a, b)$ — thing $a$ likes thing $b$.
- $Q(t)$ the value of wire Q at cycle $t$. (Assume cycles are $0, 1, 2, ...$)
- $D(t)$ the value of wire D at cycle $t$.
- Some people like peanut butter
- All people like all flavours of ice cream
- The value of Q is always equal to the value of D of the

previous cycle (if any)

## Restricting focus

Suppose we want to say that all items of a seqence of odd index are positive. We could say that

$$\forall i \in \{0, ..s.\text{length}\} \cap Odds \cdot s[i] > 0$$

where $Odds = \{k \in \mathbb{N} \mid k \bmod 2 = 1\}$. We could also express this as

$$\forall i \in \{0, ..s.\text{length}\} \cdot i \bmod 2 = 1 \Rightarrow s[i] > 0$$

Why? We have

$$(0 \bmod 2 = 1 \Rightarrow s[0] > 0) \wedge (1 \bmod 2 = 1 \Rightarrow s[1] > 0)$$
$$\wedge\, (2 \bmod 2 = 1 \Rightarrow s[2] > 0) \wedge (3 \bmod 2 = 1 \Rightarrow s[3] > 0) \wedge \cdots$$

$=$

$$(\mathsf{false} \Rightarrow s[0] > 0) \wedge (\mathsf{true} \Rightarrow s[1] > 0)$$
$$\wedge\, (\mathsf{false} \Rightarrow s[2] > 0) \wedge (\mathsf{true} \Rightarrow s[3] > 0) \wedge \cdots$$

$=$

$$\mathsf{true} \wedge (s[1] > 0) \wedge \mathsf{true} \wedge (s[3] > 0) \wedge \cdots$$

$=$

$$(s[1] > 0) \wedge (s[3] > 0) \wedge \cdots$$

Likewise, to say that at least one item of $s$ at an odd index is positive, we can write

$$\exists i \in \{0, ..s.\operatorname{length}\} \cdot i \bmod 2 = 1 \wedge s[i] > 0$$

Why?

$$\begin{aligned}
& \exists i \in \{0, ..s.\operatorname{length}\} \cdot i \bmod 2 = 1 \wedge s[i] > 0 \\
= \; & (0 \bmod 2 = 1 \wedge s[0] > 0) \vee (1 \bmod 2 = 1 \wedge s[1] > 0) \\
& \vee (2 \bmod 2 = 1 \wedge s[2] > 0) \vee (3 \bmod 2 = 1 \wedge s[3] > 0) \vee \cdots \\
= \; & \\
& (\mathsf{false} \wedge s[0] > 0) \vee (\mathsf{true} \wedge s[1] > 0) \\
& \vee (\mathsf{false} \wedge s[2] > 0) \vee (\mathsf{true} \wedge s[3] > 0) \vee \cdots \\
= \; & \\
& \mathsf{false} \vee (s[1] > 0) \vee \mathsf{false} \vee (s[3] > 0) \vee \cdots \\
= \; & \\
& (s[1] > 0) \vee (s[3] > 0) \vee \cdots
\end{aligned}$$

In general

$$\forall i \cdot P(i) \Rightarrow Q(i)$$

means, $Q$ holds for all elements in $\{i \mid P(i)\}$

And

$$\exists i \cdot P(i) \wedge Q(i)$$

means, $Q$ holds for at least one element of $\{i \mid P(i)\}$

## Example: Specifying a real-time system.

- Suppose
  - $reset(t)$ means the reset button is depressed at time $t$ and
  - $initial(t)$ means the system is in the initial state at time $t$
  - For times, we will use $\mathbb{R}^+ = \{x \in \mathbb{R} \mid x \geq 0\}$, assuming seconds as the unit.
- The system will be in the initial state at any time the reset button is pressed
- The system will be in the initial state at some time within 5 seconds of time $t$

## Nesting quantifiers

It's important to understand how quantifiers nest.

Consider the ambiguous English phrase "everybody loves somebody"

- Does it mean that there is a person whom everybody loves?
- Does it mean that, for each person, there is somebody that they love, though maybe not in each case the same person?

For the first we have

For the second

To prove the first, we can exhibit some particular person (say 'Raymond') and then show that every one loves him/her.

To prove the second, we can exhibit some particular total function $f$ and then prove that, for person $x$, $x$ loves $f(x)$.

- The system will be in the initial state within 5 seconds of the reset button being depressed:

  where $reset$ is a predicate indicating the reset button is depressed and $initial$ indicates that the system is in its initial state.
- To prove this we can exhibit a function that for each time $t$ when $reset(t)$ is true, delivers a time $u$ in the next 5 seconds such that $initial(u)$ is true.

**Nested quantifiers and games (not covered in class)**

It's easy to see the last formula as a game. We are given $reset$ and $initial$.

- Player U picks a number $t$ such that $reset(t)$ and reveals that number to player E.
- Player E then picks a number $u$ in response, such that $t \le u \le t + 5$

Player E wins if $initial(u)$. Otherwise U wins.

Now, if player E has a winning strategy, the formula true..

If player U has a winning strategy, the formula is false.

Conversely, we can understand games in terms of quantifiers.

Often chess puzzles are of the form: find a mate for white in 3 moves

For states $s$ and $t$ of a chess game, let $s \rightsquigarrow t$ mean that state $t$ can follow state $s$.

Suppose $s$ is a state where it is white's turn.

There is a guaranteed mate for white in 3 moves or fewer iff

$$\begin{aligned}
&\exists t \cdot s \rightsquigarrow t \wedge \\
&(MateForWhite(t) \\
&\vee ( \forall u \cdot t \rightsquigarrow u \Rightarrow \\
&\qquad (\exists v \cdot u \rightsquigarrow v \wedge \\
&\qquad (MateForWhite(v) \\
&\qquad \vee( \forall w \cdot v \rightsquigarrow w \Rightarrow \\
&\qquad\quad (\exists x \cdot w \rightsquigarrow x \wedge MateForWhite(x)))))))
\end{aligned}$$

Note the only free variable for this formula is $s$: it is a formula describing $s$.

More generally we can make an inductive definition.

Define $\mathit{MateIn}(1, s)$ to mean

$$\exists t \cdot s \rightsquigarrow t \wedge \mathit{MateForWhite}(t)$$

and define, for $n > 1$, $\mathit{MateIn}(n, s)$ to mean

$$\exists t \cdot s \rightsquigarrow t \wedge (\mathit{MateForWhite}(t) \vee (\forall u \cdot t \rightsquigarrow u \Rightarrow \mathit{MateIn}(n-1, u)))$$

## Relationship to set theory (not covered in class)

Recall: The notation $\{x \in S \mid \mathcal{A}\}$ means the subset of $S$ with elements $x$ such that $\mathcal{A}$ is $\mathfrak{true}$.

We can understand $\forall$ and $\exists$ in terms of set notation:

$$(\forall x \in S \cdot \mathcal{A}) = (\{x \in S \mid \mathcal{A}\} = S)$$
$$(\exists x \in S \cdot \mathcal{A}) = (\{x \in S \mid \mathcal{A}\} \neq \emptyset)$$

Therefore

| | $\forall x \in S \cdot \mathcal{A}$ | $\exists x \in S \cdot \mathcal{A}$ |
|---|---|---|
| $\emptyset = \{x \in S \mid \mathcal{A}\} = S$ | $\mathfrak{true}$ | $\mathfrak{false}$ |
| $\emptyset = \{x \in S \mid \mathcal{A}\} \subset S$ | $\mathfrak{false}$ | $\mathfrak{false}$ |
| $\emptyset \subset \{x \in S \mid \mathcal{A}\} \subset S$ | $\mathfrak{false}$ | $\mathfrak{true}$ |
| $\emptyset \subset \{x \in S \mid \mathcal{A}\} = S$ | $\mathfrak{true}$ | $\mathfrak{true}$ |

Suppose the notation $\{x \in S \cdot \mathcal{E}\}$ means the set of all values of expression $\mathcal{E}[x : y]$ where $y$ is an element of $S$. (E.g., $\{i \in \mathbb{N} \cdot 2i\} = \{0, 2, 4, ...\}$)

We can understand $\forall$ and $\exists$ in as follows

$$(\forall x \in S \cdot \mathcal{A}) = (\mathfrak{false} \notin \{x \in S \cdot \mathcal{A}\})$$
$$(\exists x \in S \cdot \mathcal{A}) = (\mathfrak{true} \in \{x \in S \cdot \mathcal{A}\})$$

Since $\mathcal{A}$ is boolean, the set $\{x \in S \cdot \mathcal{A}\}$ can have four values (if well defined)

| $\{x \in S \cdot \mathcal{A}\}$ | $\forall x \in S \cdot \mathcal{A}$ | $\exists x \in S \cdot \mathcal{A}$ |
|---|---|---|
| $\emptyset$ | $\mathfrak{true}$ | $\mathfrak{false}$ |
| $\{\mathfrak{false}\}$ | $\mathfrak{false}$ | $\mathfrak{false}$ |
| $\{\mathfrak{true}\}$ | $\mathfrak{true}$ | $\mathfrak{true}$ |
| $\{\mathfrak{false}, \mathfrak{true}\}$ | $\mathfrak{false}$ | $\mathfrak{true}$ |

## Alternative Notations (not covered in class)

It is quite common to leave out the $\in S$ part when the domain of the variable is understood by some other means.

Many writers also leave out the $\cdot$ or replace it by some other symbol. There is a wide variety in conventions for using parentheses. So you may see any of

$$\forall \mathcal{V} \cdot \mathcal{A}$$
$$\forall \mathcal{V}\, \mathcal{A}$$
$$\forall \mathcal{V}(\mathcal{A})$$
$$\forall \mathcal{V}, \mathcal{A}$$
$$[\forall \mathcal{V} \in S,\ \mathcal{A}]$$
$$\forall \mathcal{V}[\mathcal{A}]$$
$$(\mathcal{V})\mathcal{A}$$

## Quantifier Laws (not covered in class)

There are a number of laws of predicate calculus which are useful to know.

These are some.

Identity laws:

$$\begin{aligned}(\forall x \in S \cdot \mathfrak{true}) &= \mathfrak{true}\\ (\exists x \in S \cdot \mathfrak{false}) &= \mathfrak{false}\\ (\forall x \in S \cdot \mathfrak{false}) &= \mathfrak{false}\text{, provided } S \neq \emptyset\\ (\exists x \in S \cdot \mathfrak{true}) &= \mathfrak{true}\text{, provided } S \neq \emptyset\\ (\forall x \in \emptyset \cdot \mathcal{A}) &= \mathfrak{true}\\ (\exists x \in \emptyset \cdot \mathcal{A}) &= \mathfrak{false}\end{aligned}$$

Change of variable: Provided $y$ does not occur free in $\mathcal{A}$,

$$\begin{aligned}(\forall x \in \mathbb{N} \cdot \mathcal{A}) &= (\forall y \in \mathbb{N} \cdot \mathcal{A}[x : y])\\ (\exists x \in \mathbb{N} \cdot \mathcal{A}) &= (\exists y \in \mathbb{N} \cdot \mathcal{A}[x : y])\end{aligned}$$

De Morgan's laws

$$\begin{aligned}(\forall x \in S \cdot \mathcal{A}) = \neg(\exists x \in S \cdot \neg\mathcal{A})\\ (\exists x \in S \cdot \mathcal{A}) = \neg(\forall x \in S \cdot \neg\mathcal{A})\end{aligned}$$

Domain splitting

$$\begin{aligned}(\forall x \in S \cup T \cdot \mathcal{A}) &= (\forall x \in S \cdot \mathcal{A}) \wedge (\forall x \in T \cdot \mathcal{A})\\ (\exists x \in S \cup T \cdot \mathcal{A}) &= (\exists x \in S \cdot \mathcal{A}) \vee (\exists x \in T \cdot \mathcal{A})\end{aligned}$$

Splitting

$$\begin{aligned}(\forall x \in S \cdot \mathcal{A} \wedge \mathcal{B}) &= (\forall x \in S \cdot \mathcal{A}) \wedge (\forall x \in S \cdot \mathcal{B})\\ (\exists x \in S \cdot \mathcal{A} \vee \mathcal{B}) &= (\exists x \in S \cdot \mathcal{A}) \vee (\exists x \in S \cdot \mathcal{B})\end{aligned}$$

Trading

$$(\forall x \in S \cdot \mathcal{A} \Rightarrow \mathcal{B}) = (\forall x \in \{x \in S \mid \mathcal{A}\} \cdot \mathcal{B})$$
$$(\exists x \in S \cdot \mathcal{A} \wedge \mathcal{B}) = (\exists x \in \{x \in S \mid \mathcal{A}\} \cdot \mathcal{B})$$

One-point laws: Provided $x$ does not appear free in $\mathcal{F}$ and that $\mathcal{F} \in S$,

$$(\forall x \in S \cdot (x = \mathcal{F}) \Rightarrow \mathcal{A}) \;=\; \mathcal{A}[x : \mathcal{F}]$$
$$(\exists x \in S \cdot (x = \mathcal{F}) \wedge \mathcal{A}) \;=\; \mathcal{A}[x : \mathcal{F}]$$

Commutative: Provided $x$ is not free in $T$ and $y$ is not free in $S$,

$$(\forall x \in S \cdot \forall y \in T \cdot \mathcal{A}) \;=\; (\forall y \in T \cdot \forall x \in S \cdot \mathcal{A})$$
$$(\exists x \in S \cdot \exists y \in T \cdot \mathcal{A}) \;=\; (\exists y \in T \cdot \exists x \in S \cdot \mathcal{A})$$

Distributive laws: Provided $x$ is not free in $\mathcal{A}$

$$\mathcal{A} \wedge (\exists x \in S \cdot \mathcal{B}) \;=\; (\exists x \in S \cdot \mathcal{A} \wedge \mathcal{B})$$
$$\mathcal{A} \vee (\forall x \in S \cdot \mathcal{B}) \;=\; (\forall x \in S \cdot \mathcal{A} \vee \mathcal{B})$$
$$(\mathcal{A} \Rightarrow (\forall x \in S \cdot \mathcal{B})) \;=\; (\forall x \in S \cdot \mathcal{A} \Rightarrow \mathcal{B})$$

Distributive laws: Provided $S \neq \emptyset$ and $x$ is not free in $\mathcal{A}$

$$(\mathcal{A} \wedge (\forall x \in S \cdot \mathcal{B})) \;=\; (\forall x \in S \cdot \mathcal{A} \wedge \mathcal{B})$$
$$(\mathcal{A} \vee (\exists x \in S \cdot \mathcal{B})) \;=\; (\exists x \in S \cdot \mathcal{A} \vee \mathcal{B})$$
$$(\mathcal{A} \Rightarrow (\exists x \in S \cdot \mathcal{B})) \;=\; (\exists x \in S \cdot \mathcal{A} \Rightarrow \mathcal{B})$$

## Precedence and associativity

As you know, mathematics uses “precedence conventions” to reduce the need for parentheses. For example we all know that

$$w \times x + y \times z$$

means

$$(w \times x) + (y \times z)$$

rather than

$$w \times (x + y) \times z$$

as $\times$ has “higher” precedence than $+$.

Furthermore we know that

$$a - b + c \text{ means } (a - b) + c$$

rather than $a - (b + c)$ as $-$ and $+$ are “left associative”.

Some operators are associative meaning it doesn’t matter how we add parentheses. E.g.

$$((a \wedge b) \wedge c) = (a \wedge b \wedge c) = (a \wedge (b \wedge c))$$

On the other hand

$$a \leq b < c \text{ means } (a \leq b) \wedge (b < c)$$

and we say that $\leq$, $<$, $=$, etc are “chaining”

The following table shows many of the operators used in the course in order of precedence (highest to lowest)

| | |
|---|---|
| $x(y)$ $E[v : F]$ | LA |
| $-x$ $\neg x$ | |
| $x \times y$ $x/y$ | LA |
| $x + y$ $x - y$ | LA |
| $\cap$ | A |
| $\cup$ | A |
| $x = y$ $x \leq y$ $x < y$ $x \in y$ | Ch |
| $x \wedge y$ | A |
| $x \vee y$ | A |
| $x \Rightarrow y$ NA $x \Leftrightarrow y$ $x \not\Leftrightarrow y$ | A |
| $\forall v \in S \cdot x$ $\exists v \in S \cdot x$ | |

where

| | |
|---|---|
| LA | Left associative |
| RA | Right associative |
| A | Associative |
| NA | Nonassociative |
| Ch | Chaining |

The low precedence of the quantifiers basically means that the scope of a quantified variable extends to the right to the end of the formula, unless there is explicit parenthesization or punctuation to stop it. I recommend putting quantifications in parentheses except when there is no possible confusion.

That $\wedge$ has higher precedence than $\vee$ is conventional, but I recommend using extra parentheses, e.g. to write

$p \wedge q \vee r$ as $(p \wedge q) \vee r$