UECM1304 Discrete Mathematics with Applications
Topic 4: Relations and Applications

Lecturer: Dr. Liew How Hui
Email: liewhh@utar.edu.my

10 hours

Sets, relations and functions are how mathematician organised mathematical objects. In Topic 1 and Topic 2, we have a set of statements Prop, a set of satisfiable formulas SAT, etc. In Topic 3, we have learned to perform proving mathematical statements involving a set of natural numbers $\mathbb{N}$, a set of integers $\mathbb{Z}$, a set of rationales $\mathbb{Q}$ and/or a set of real numbers $\mathbb{R}$.

In this topic, we will explore properties of sets and relations as well as the generalisation of equality and order called the equivalence relation and partial order relation.

In computer science, the denotational semantics theory builds "models" for (programming) languages using sets/types and functions.

Contents

CLO4: Express relations correctly with their mathematical properties

§4.1 Sets and Relations

Sets are used in mathematics to group elements (without repetition) of the "same property" (many of them can be characterised by predicates) and relations are used to relate multiple sets and one set.

There are little characterisations for relations on multiple sets but equivalence relations and partial order relations are two important classes of relations on one set.

§4.1.1 Definitions

Informally, a **set** is any well-defined list, collection, or class of objects. The objects comprising the set are called its **elements** or **members**. The statement "p is an element of a set A" is written as "$p \in A$".

The operations associated with sets A and B are defined below.

Definition 4.1.1. 1. A is a ***subset*** of B, denoted by $A \subset B$, if $x \in A$ then $x \in B$.

2. $\mathscr{P}(A)$ is a ***power set*** of A if it is the family of all the subsets of any set A.

Remark 4.1.2. Some books use $A \subseteq B$ to denote A is a subset of B. However, in this course, $A \subset B$ will be used.

Example 4.1.3. Given $A = \{1, 2, 3\}$.
$\emptyset$ is a subset of A, i.e. $\emptyset \subset A$.
A is a subset of A, i.e. $A \subset A$.
A is a subset of $\mathbb{N}$, i.e. $A \subset \mathbb{N}$.
$\mathscr{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$.

Definition 4.1.4. 1. The ***union*** of A and B,

$$A \cup B := \{x \ : \ x \in A \vee x \in B\}.$$

2. The ***intersection*** of A and B,

$$A \cap B := \{x \ : \ x \in A \wedge x \in B\}$$

3. The ***difference*** of A and B,

$$A \setminus B := \{x \ : \ x \in A \wedge x \notin B\}$$

4. The ***(Cartesian) product*** of A and B,

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

where (a, b) are ordered pairs of A and B.

5. The product can be generalised to the case of n sets $A_1, \cdots, A_n$, i.e. the ***n-ary Cartesian product over sets*** $A_1, \cdots, A_n$,

$$A_1 \times \cdots \times A_n := (((A_1 \times A_2) \times A_3) \times \cdots A_n) = \{((x_1, x_2), \cdots, x_n) : x_i \in A_i\}.$$

Example 4.1.5. Given $A = \{1, 2, 3\}$, $B = \{1, a, b\}$.
$A \cup B = \{1, 2, 3, a, b\}$.
$A \cap B = \{1\}$.
$A \setminus B = \{2, 3\}$.
$B \setminus A = \{a, b\}$.
$A \times B = \{(1, 1), (1, a), (1, b), (2, 1), (2, a), (2, b), (3, 1), (3, a), (3, b)\}$.

Example 4.1.6. Let A and B be two sets. Show that $A \times B = \emptyset$ iff $A = \emptyset$ or $B = \emptyset$.

We have not shown the proof for "iff" in Topic 3, this is the first "iff" example.

Proof:
$\Rightarrow$: (Contrapositive Proof) If $A \neq \emptyset$ and $B \neq \emptyset$, $\exists x \in A$ and $\exists y \in B$ such that $(x, y) \in A \times B$. This implies $A \times B \neq \emptyset$.

$\Leftarrow$: (Contrapositive Proof) If $A \times B \neq \emptyset$, then $\exists z \in A \times B$, $z = (z_1, z_2)$, $z_1 \in A$ and $z_2 \in B$, hence $A \neq \emptyset$ and $B \neq \emptyset$. $\square$

Definition 4.1.7. Let A and B be two nonempty sets. A ***(binary) relation from*** A ***to*** B, R, is a subset of $A \times B$, i.e. $R \subset A \times B$. In particular, when $B = A$, we have a ***(binary) relation on*** A, R, i.e. $R \subset A \times A$.

When $(a, b) \in R$, a ***is related to*** b ***by*** R, and can be written as aRb

If a ***is not related to*** b ***by*** R, we denote it as $(a, b) \notin R$ or $a \not R b$.

The ***domain of*** R and ***range of*** R are defined respectively as

$$\text{Dom}(R) = \{a \in A \ : \ \exists b \in B, aRb\},$$
$$\text{Range}(R) = \{b \in B \ : \ \exists a \in A, aRb\}.$$

Example 4.1.8. The domain and range of the relation

$$R = \{(a, a), (a, b), (b, b), (b, c), (d, c)(d, e)\}$$

is

$$\text{Dom}(R) = \{a, b, d\}, \quad \text{Range}(R) = \{a, b, c, e\}.$$

Example 4.1.9. Given $A = \{-3, -2, -1, 0, 1, 2, 3, 4\}$, $B = \{0, 1, 4\}$ and define the relation R from A to B as $R = \{(x, y) : y = x^2\}$. Determine the domain and range of R.

Example 4.1.10. Let $A = \{3, 4, 5, 6, 7\}$ and let R be the relation $<$ (less than) on A. List the elements of R.

Example 4.1.11. Let R be a relation on $A = \{1, 2, 3, 4\}$ defined by: $xRy = x$ divides y and $x < y$. List all elements in R.

Definition 4.1.12 (`https://en.wikipedia.org/wiki/Finitary_relation`)**.** R is an n**-ary relation** or a **finitary relation** over a sequence of sets A_1, ..., A_n if it is a subset of the Cartesian product $A_1 \times \cdots \times A_n$, i.e. $R \subset A_1 \times \cdots \times A_n$.

Definition 4.1.13. A ***function from*** A ***to*** B, f, is a relation from A to B such that for every $x \in \text{Dom}(f)$, there is a **unique** value y such that xfy or what we usually write $y = f(x)$. It is normally written as

$$f : A \to B, \quad x \mapsto y.$$

A function f is called a ***bijection*** if $\text{Dom}(f) = A$ and for every $y \in B$, there is a unique $x \in A$ such that $f(x) = y$.

A function f is called an ***injection*** if for every x_1 and x_2 in A, if $f(x_1) = f(x_2)$ then $x_1 = x_2$.

A function f is called an ***surjection*** if for every y in B, there is an x in A such that $y = f(x)$. In other words, $\text{Range}(f) = B$.

Definition 4.1.14. Let $X : I \to S$ be a function from the ***index set*** I to the set of sets S. The function X is called a ***family of sets***, which is normally denoted as $\{X_i := X(i) | i \in I\}$ or $(X_i)_{i\in I}$.

Using logic and the notion of function/family defined in Definition 4.1.14, one can generalise the set operations in Definition 4.1.4 as follows.

Definition 4.1.15. Let $(X_i)_{i\in I}$ be a family of subsets of a set E. Then its

1. **union**, $\bigcup_{i\in I} X_i := \{x \in E \ : \ \exists i(i \in I \wedge x \in X_i)\}$.
2. **intersection**, $\bigcap_{i\in I} X_i := \{x \in E \ : \ \forall i(i \in I \to x \in X_i)\}$.
3. **complement**, $X_i^c := \{x \in E \ : \ x \notin X_i\}$, $i \in I$.
4. **product**, $\prod_{i\in I} X_i := \{f : I \to \bigcup_{i\in I} X_i \ : \ f(i) \in X_i\}$.

Remark 4.1.16. Note that when the index set I is empty, we have the following:

$$\bigcup_{i\in I} X_i = \emptyset, \quad \bigcap_{i\in I} X_i = E, \quad \prod_{i\in I} X_i = \{\emptyset\}.$$

§4.1.2 Properties of Relations

There is very little mathematical characterisation about n-ary relation. In computer science, it is common to refer to a Boolean valued function as an **n-ary predicate**. In formal logic, the relation R constitutes a "model" or a "relational structure" or an "interpretation" for some n-ary predicate symbol.

The only relation that have good mathematical characterisation is the **binary relation on a set**. This is because we can generalise the equality relation and the ordering relation.

Definition 4.1.17. A relation R on a set A is called ***reflexive*** if $(a, a) \in R$ for every $a \in A$.

A relation R on a set A is called ***irreflexive*** if $(a, a) \notin R$ for every $a \in A$.

Example 4.1.18. Let $A = \{1, 2, 3, 4, 5, 6\}$. Determine whether the following relations are reflexive, irreflexive or neither.

1. $R_1 = \{(1, 1), (1, 2), (2, 2), (3, 3), (4, 4), (5, 5), (5, 6), (6, 6)\}$

   Solution: R_1 is reflexive since $(a, a) \in R_1$ for all $a \in A$. But it is not irreflexive.

2. $R_2 = \{(x, y) \in A \times A | x > y\}$

3. $R_3 = \{(1, 1), (1, 3), (2, 2)\}$

Definition 4.1.19. Given a relation R on a set A.

1. R is called ***symmetric*** if when $(a, b) \in R$, $(b, a) \in R$.
2. R is called ***asymmetric*** if when $(a, b) \in R$, $(b, a) \notin R$.
3. R is called ***antisymmetric*** if when $(a, b) \in R$ and $(b, a) \in R$, $a = b$.

Remark 4.1.20. Beware that not symmetric does not mean asymmetric or antisymmetric. However, asymmetric implies antisymmetric but not vice versa.

Example 4.1.21. Let $A = \{1, 2, 3, 4, 5, 6\}$. Determine whether the following relations are symmetric, asymmetric and anti-symmetric.

- $R_0 = \emptyset$

Solution: By definition, since R_0 does not have any element, $(a, b) \in R_0$ is always false, it satisfies all rules. It is symmetric, asymmetric and anti-symmetric.

- $R_1 = \{(1, 2), (2, 1), (2, 3), (3, 2), (3, 4), (5, 5)\}$.

Solution: R_1 is *not symmetric* since $(3, 4) \in R_1$ but $(4, 3) \notin R_1$.

R_1 is *not asymmetric* since $(1, 2) \in R_1$ and $(2, 1) \in R_1$ but $1 \neq 2$.

R_1 is *not antisymmetric* since $1 \neq 2$ but $(1, 2) \in R_1$ and $(2, 1) \in R_1$.

- $R_2 = \{(1, 2), (2, 1), (2, 4), (3, 4), (4, 2), (4, 3), (5, 6), (6, 5), (6, 6)\}$.

- $R_3 = \{(a, b) \in A \times A : a > b\}$.

Definition 4.1.22. A relation on R on a set A is ***transitive*** if whenever $(a, b) \in R$ and $(b, c) \in R$, then $(a, c) \in R$.

A relation R on a set A is ***not transitive*** if there exist $a, b, c \in A$ such that $(a, b) \in R$ and $(b, c) \in R$ but $(a, c) \notin R$.

Example 4.1.23. $<, \leq, >, \geq$ are transitive relations on $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ because we have

- if $a < b$, $b < c$, then $a < c$
- if $a \leq b$, $b \leq c$, then $a \leq c$
- if $a > b$, $b > c$, then $a > c$
- if $a \geq b$, $b \geq c$, then $a \geq c$

§4.1.3 Application of Sets and Relations

The following are some areas where sets and relations are applied:

- The relation R, a subset of $A_1 \times \cdots \times A_n$ (Definition 4.1.4), is a mathematical representation of predicate $P(x_1, \cdots, x_n)$ as in Topic 1's Definition 1.8.1 (terms and formulas).
- Some specification languages such as Z notation (uses a mixed of sets and types?), TLA+ language (for concurrent and distributed programs) use sets in **specification** and **modelling**.
- Denotational semantics (Section 4.5.3, also known as *mathematical semantics* or *Scott-Strachey semantics*) is an approach for formalising the "meanings" of programming languages by constructing mathematical objects (called denotations, e.g. domains, a special kinds of partial ordered set (Section 4.5)) that describe the meanings of the expressions from programming languages.
- Relational programming with `https://minikanren.org`: Set relations allows programming language to deal with predicates rather than just functions.

Example 4.1.24 (TLA+ language)**.** One can find many TLA+ examples from `https://github.com/tlaplus/Examples`. The simplest illustration of the application of TLA+ is `https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem` which considers a situation where many threads are reading and writing a shared resource but no thread may access (read or write) the shared resource when another thread is in the act of writing to it. The specifications of the problem in TLA+ language are listed below.

ReadersWriters/MC.tla

```
1  ------------------------- MODULE ReadersWriters -------------------------
2  (***************************************************************************)
3  (* This solution to the readers-writers problem, cf.                       *)
4  (* https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem,          *)
5  (* uses a queue in order to fairly serve all requests.                     *)
6  (***************************************************************************)
7  EXTENDS FiniteSets, Naturals, Sequences
8
9  CONSTANT NumActors
10
11 VARIABLES
12     readers, \* set of processes currently reading
13     writers, \* set of processes currently writing
14     waiting  \* queue of processes waiting to access the resource
15
16 vars == <<readers, writers, waiting>>
17
18 Actors == 1..NumActors
19
20 ToSet(s) == { s[i] : i \in DOMAIN s }
21
22 read(s)  == s[1] = "read"
23 write(s) == s[1] = "write"
24
25 WaitingToRead  == { p[2] : p \in ToSet(SelectSeq(waiting, read)) }
26
27 WaitingToWrite == { p[2] : p \in ToSet(SelectSeq(waiting, write)) }
28
29 ---------------------------------------------------------------------------
30
31 (***********)
32 (* Actions *)
33 (***********)
34
35 TryRead(actor) ==
36     /\ actor \notin WaitingToRead
37     /\ waiting' = Append(waiting, <<"read", actor>>)
38     /\ UNCHANGED <<readers, writers>>
39
40 TryWrite(actor) ==
41     /\ actor \notin WaitingToWrite
42     /\ waiting' = Append(waiting, <<"write", actor>>)
43     /\ UNCHANGED <<readers, writers>>
44
45 Read(actor) ==
46     /\ readers' = readers \union {actor}
47     /\ waiting' = Tail(waiting)
48     /\ UNCHANGED writers
49
```

```
Write(actor) ==
    /\ readers = {}
    /\ writers' = writers \union {actor}
    /\ waiting' = Tail(waiting)
    /\ UNCHANGED readers

ReadOrWrite ==
    /\ waiting /= <<>>
    /\ writers = {}
    /\ LET pair  == Head(waiting)
           actor == pair[2]
       IN CASE pair[1] = "read" -> Read(actor)
            [] pair[1] = "write" -> Write(actor)

StopActivity(actor) ==
    IF actor \in readers
    THEN /\ readers' = readers \ {actor}
         /\ UNCHANGED <<writers, waiting>>
    ELSE /\ writers' = writers \ {actor}
         /\ UNCHANGED <<readers, waiting>>

Stop == \E actor \in readers \cup writers : StopActivity(actor)

----------------------------------------------------------------------------

(*****************)
(* Specification *)
(*****************)

Init ==
    /\ readers = {}
    /\ writers = {}
    /\ waiting = <<>>

Next ==
    \/ \E actor \in Actors : TryRead(actor)
    \/ \E actor \in Actors : TryWrite(actor)
    \/ ReadOrWrite
    \/ Stop

Fairness ==
    /\ \A actor \in Actors : WF_vars(TryRead(actor))
    /\ \A actor \in Actors : WF_vars(TryWrite(actor))
    /\ WF_vars(ReadOrWrite)
    /\ WF_vars(Stop)

Spec == Init /\ [][Next]_vars /\ Fairness

----------------------------------------------------------------------------

(**************)
(* Invariants *)
(**************)

TypeOK ==
    /\ readers \subseteq Actors
```

```
106     /\ writers \subseteq Actors
107     /\ waiting \in Seq({"read", "write"} \times Actors)
108
109 Safety ==
110     /\ ~(readers /= {} /\ writers /= {})
111     /\ Cardinality(writers) <= 1
112
113 (**************)
114 (* Properties *)
115 (**************)
116
117 Liveness ==
118     /\ \A actor \in Actors : []<>(actor \in readers)
119     /\ \A actor \in Actors : []<>(actor \in writers)
120     /\ \A actor \in Actors : []<>(actor \notin readers)
121     /\ \A actor \in Actors : []<>(actor \notin writers)
122
123 ============================================================================
```

We can observe the use of set at lines 20, 25, 27, 36, 41, etc. and the use of logic from line 35 to line 121.

To model the problem, one needs to specify the `NumActors` in the configuration file `MC.cfg` and then specific the value in `MC.tla`.

ReadersWriters/MC.cfg

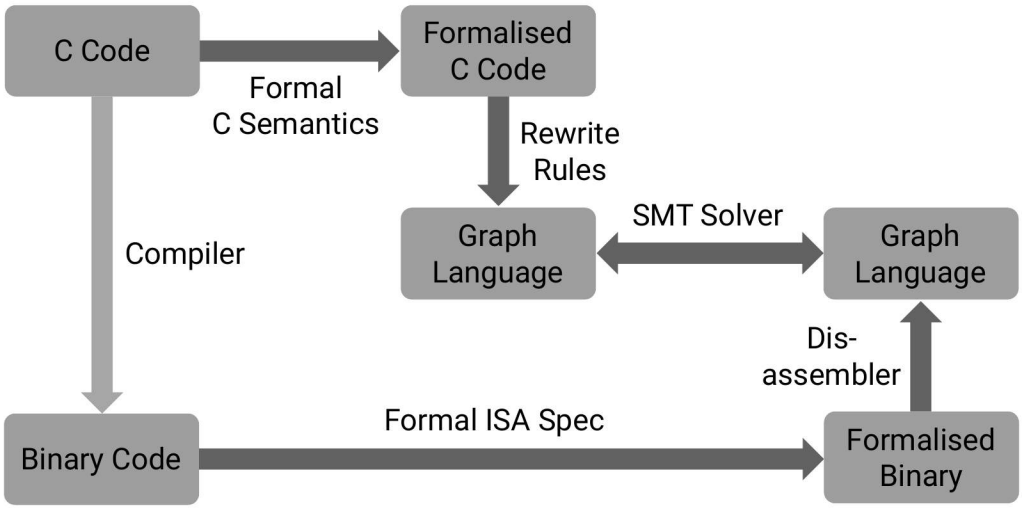
```
CONSTANT
    NumActors <- n

SPECIFICATION
    Spec

INVARIANT
    TypeOK
    Safety

PROPERTY
    Liveness
```

ReadersWriters/MC.tla

```
---- MODULE MC ----

EXTENDS ReadersWriters, TLC

n == 3

\* The following invariants are all violated:
\* - Cardinality(readers) < n
\* - Cardinality(WaitingToWrite) < n
\* - WaitingToRead /= {} => WaitingToWrite = {}
\* - WaitingToWrite /= {} => WaitingToRead = {}

==================
```

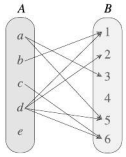
As pointed out in `https://lamport.azurewebsites.net/tla/tools.html`, the Apalache Symbolic Model Checker for TLA+ (`https://apalache-mc.org`, implemented in Scala language) translates TLA+ into the logic supported by SMT solvers (such as Z3).

A real-world application of SMT solver is given in seL4 operating system for embedded system Heiser [2025] shown in the following figure.



§4.2 Representing Relations: Boolean Matrix and Digraph

To make binary relations easier to understand, various forms of representations are proposed:

- **Listing Tuples (Roster Notation)**: E.g. $R = \{(x_1, y_1), ..., (x_n, y_n)\}$.
  This is useful when the domain and range are finite.
- **Arrow Diagram Representation**: E.g. 

  This is useful when the domain and range are finite.
- **(Boolean/Logical) Matrix Representation**: Section 4.2.1
  This is useful when the domain and range are the same and finite.
- **Directed Graph (Digraph) Representation**: Section 4.2.2
  This is useful when the domain and range are the same and finite.
- **Set Builder Notation**: E.g. $\{(x, y) : y = x^2\}$
  This is useful when the mathematical formulation is clear.
- **Cartesian Plane Graph**:
  This is useful for visualisation for domain and range which are numbers.

§4.2.1 (Boolean/Logical) Matrix

A relation from a finite set A to a finite set B has a "Boolean" (or binary or logical) matrix representation.

Definition 4.2.1. Let $A = \{a_1, a_2, \cdots, a_m\}$ and $B = \{b_1, b_2, \cdots, b_n\}$ and let R be a relation from A to B. Then R can be represented by an $m \times n$ **Boolean matrix** or **logical matrix** or **0-1 integral matrix**

$$M_R = [m_{ij}]$$

called the ***matrix representation of*** R, where

$$m_{ij} = \begin{cases} 1 \text{ or T}, & \text{if } (a_i, b_j) \in R, \\ 0 \text{ or F}, & \text{if } (a_i, b_j) \notin R. \end{cases}$$

We can regard the Boolean matrix as usual matrix and apply the usual matrix multiplication or define matrix operations specific for logical operations generalising from the logical connectives $\wedge$ and $\vee$.

Definition 4.2.2. Let $M_R = [r_{ij}]$ and $M_S = [s_{ij}]$ be two $m \times n$ Boolean matrices. Let $M_T = [t_{jk}]$ be an $n \times k$ Boolean matrix.

- The ***meet*** (Conjunction or AND) of M_R and M_S, $M_R \wedge M_S = [r_{ij} \wedge s_{ij}]$;
- The ***join*** (Disjunction or OR) of M_R and M_S, $M_R \vee M_S = [r_{ij} \vee s_{ij}]$;
- The ***product*** of M_R and M_T, $M_R \odot M_T = [\bigvee_j (r_{ij} \wedge t_{jk})]$

Note that $0 \wedge 0 = 0$, $0 \wedge 1 = 0$, $1 \wedge 0 = 0$, $1 \wedge 1 = 1$; $0 \vee 0 = 0$, $0 \vee 1 = 1$, $1 \vee 0 = 1$, $1 \vee 1 = 1$. We can regard $0 = \text{F}$, $1 = \text{T}$.

Theorem 4.2.3. *Let A and B be defined in Definition 4.2.1 and $C = \{c_1, c_2, \cdots, c_k\}$. Let $R_1, R_2 \subset A \times B$, $R_3 \subset B \times C$ and the matrix representations of R_i be M_{R_i}, $i = 1, 2, 3$.*

- *The matrix representation of* $R_1 \cap R_2$, $M_{R_1 \cap R_2} = M_{R_1} \wedge M_{R_2}$;
- *The matrix representation of* $R_1 \cup R_2$, $M_{R_1 \cup R_2} = M_{R_1} \vee M_{R_2}$;
- *The matrix representation of* $R = \{(a, c) \mid \exists b\,(aR_1b \wedge bR_3c)\} \subset A \times C$, $M_R = M_{R_1} \odot M_{R_3}$.

Example 4.2.4. Let $A = \{a, b, c, d, e, f\}$, $B = \{1, 2, 4, 5, 6\}$ and let $R = \{(a, 2), (a, 5), (b, 5), (b, 6), (c, 1), (c, 4), (e, 5), (f, 6)\}$ be the relation from A to B. Find the matrix representation of R.

Solution: $M_R = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$

Example 4.2.5. Let $M_R = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$ be the matrix representation of a relation R from $A = \{a, b, c\}$ to $B = \{1, 2, 3, 4\}$. Write down the set R.

In the following we will limit ourselves to the discussion of (binary) relations on a set. For a relation on a set, it can be represented by a square matrix.

Example 4.2.6. Let $A = \{1, 2, 3, 4, 5\}$ and $R = \{(1, 1), (1, 2), (2, 1), (3, 4), (3, 5), (4, 5)\}$. Write down the matrix representation M_R of R.

Solution: The matrix representation $M_R = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

Square Matrix in Standard ML (`warshall_mat.sml`)

```
(* Integral Matrix Represetation of a Relation on n-Element Set *)
signature SQUARE_ARRAY =
sig
  type 'a squareArray
  val new : int * 'a -> 'a squareArray
  val get : 'a squareArray -> int * int -> 'a
  val set : 'a squareArray -> int * int -> 'a -> unit
end;

structure SquareArray : SQUARE_ARRAY =
struct
  type 'a squareArray = int * 'a array
  fun new (n, fill)       = (n, Array.array (n * n, fill))
  fun get (n, r) (i, j)   = Array.sub    (r, (i - 1) + (n * (j - 1)))
  fun set (n, r) (i, j) x = Array.update (r, (i - 1) + (n * (j - 1)), x)
```

```
end;

(* Example 4.2.4 *)
val M_R = SquareArray.new (5, 0);
val _ = SquareArray.set M_R (1,1) 1
val _ = SquareArray.set M_R (1,2) 1
val _ = SquareArray.set M_R (2,1) 1
val _ = SquareArray.set M_R (3,4) 1
val _ = SquareArray.set M_R (3,5) 1
val _ = SquareArray.set M_R (4,5) 1
(* Print the Boolean matrix *)
val _ = Array.tabulate (5,
          fn i => (Array.tabulate (5,
            fn j => print (Int.toString (SquareArray.get M_R (i+1,j+1)) ^ " ")),
          print "\n"));
```

Remark 4.2.7. Given a relation R on a finite set A.

- R is **reflexive** iff ***the diagonal element of matrix representation*** M_R ***are all 1***.
- R is **irreflexive** iff ***the diagonal element of*** M_R ***are all 0***.
- R is **neither reflexive nor irreflexive** iff ***0's and 1's co-exist on the main diagonal of*** M_R.
- R is **symmetric** iff M_R ***is a symmetric matrix***, i.e. $M_R = M_R^T$.
- R is **asymmetric** iff M_R ***has no symmetric pair*** and its ***diagonal contains all 0's***.
- R is **antisymmetric** iff M_R ***has no symmetric pair***.
- R is **transitive** iff M_R ***has the property if*** $m_{ij} = 1$ ***and*** $m_{jk} = 1$ ***then*** $m_{ik} = 1$, i.e. if every non-zero element in the matrix M_R^2 corresponds to a non-zero element in M_R, then R is transitive.

Example 4.2.8. Let $A = \{1, 2, 3, 4, 5\}$ and R_i, $i = 1, 2, 3, 4$ be relations on A. Determine whether the following relations are reflexive, irreflexive, symmetric, anti-symmetric and transitive.

- $R_1 = \{(1, 2), (3, 4)\}$

  Solution: R_1 is not reflexive because $(1, 1) \notin R_1$.
  R_1 is irreflexive because the diagonal elements are all zeros.
  R_1 is not symmetric because $(1, 2) \in R_1$ but $(2, 1) \notin R_1$.
  R_1 is antisymmetric because there is no symmetric pair.
  R_1 is transitive since there are no elements a, b and c in A such that $(a, b) \in R_1$ and $(b, c) \in R_1$, but $(a, c) \notin R_3$. We can also show this by comparing $M_{R_1}^2$ to M_{R_1}:

$$M_{R_1} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad M_{R_1}^2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

  Since there is no non-zero element in $M_{R_1}^2$, the rule in Remark 4.2.7 is satisfied, R_1 is transitive.

- $R_2 = \{(1, 1), (1, 2), (2, 1), (3, 4), (3, 5), (4, 5)\}$

Solution:
R_2 is not reflexive because $(5,5) \notin R_2$.
R_2 is not irreflexive because $(1,1) \notin R_2$.
R_2 is not symmetric because $(3,4) \in R_2$ but $(4,3) \notin R_2$.
R_2 is not antisymmetric because we have a symmetric pair $(1,2) \in R_2$ and $(2,1) \in R_2$.
R_2 is *not transitive* since $(2,1) \in R_2$ and $(1,2) \in R_2$ but $(2,2) \notin R_2$.

- $R_3 = \{(1,2),(1,3),(1,4),(4,4)\}$

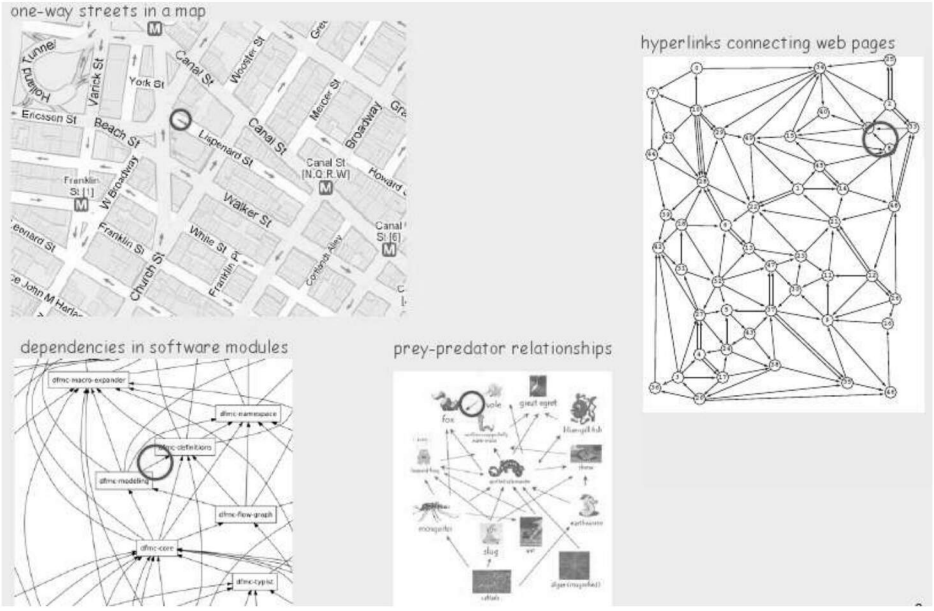
- $R_4 = \{(1,2),(2,1),(2,2),(2,3),(3,2)\}$

§4.2.2 Digraphs

The digraph representation for the binary relation from A to B is the arrow diagram representation (or bipartite graph (`https://en.wikipedia.org/wiki/Bipartite_graph`) representation) mentioned earlier.

When $A = B = S$, a relation $R \subset S \times S$ becomes a binary relation on the set S. We can use a **directed graph (or digraph) representation** for R which is a pair (S, R) where S is the set of **nodes** and R is the set of **edges**.

Digraph is an abstraction of oriented connections of objects, such as various kinds of maps with directions shown below.



The object is marked as a node while the connection is marked as an edge in a digraph. Some applications of digraph are shown in the table below:

digraph	node	edge
financial	stock, currency	transaction
transportation	street intersection, airport	highway, airway route
scheduling	task	precedence constraint
WordNet	synset	hypernym
Web	web page	hyperlink
game	board position	legal move
telephone	person	placed call
food web	species	predator-prey relation
infectious disease	person	infection
citation	journal article	citation
object graph	object	pointer
inheritance hierarchy	class	inherits from
control flow	code block	jump

Definition 4.2.9. A ***directed graph*** or ***digraph*** is a pair $G = (V, E)$ such that

- V is a set whose elements are called ***nodes*** or ***vertices***;
- E is a set of ordered pairs of nodes, called ***(directed) edges***, ***arcs*** or ***arrows***.

Definition 4.2.10. The ***adjacency matrix*** of a digraph $G = (V, E)$ (with loops and multiple arcs) is the integer-valued matrix with rows and columns corresponding to the digraph nodes, where a non-diagonal

entry a_{ij} is the number of arcs from node i to node j, and the diagonal entry a_{ii} is the number of loops at node i. The adjacency matrix for a digraph is unique up to the permutations of rows and columns. It is denoted $Adj(G)$.

The digraph representation of a relation R on a set A is defined below.

Definition 4.2.11. If R is a relation on a finite set A, we can construct a ***digraph representation of*** R, denoted by G_R, as follows:

1. Draw a small circle for each element of A and label the circle with the corresponding element of A. Such a circle is the node or vertex of the graph.

2. Draw a line with an arrow from a node v_i to a node v_j iff $v_i R v_j$. This line (v_i, v_j) is the edge when $i \neq j$ and it is a loop when $i = j$.

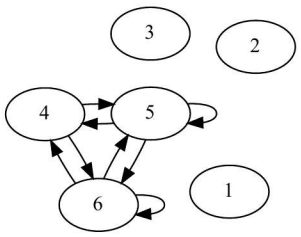
Remark 4.2.12. The matrix representation M_R of a relation R on a set A is equivalent to the adjacency matrix of the digraph representation G_R.

Example 4.2.13. Let $A = \{1, 2, 4, 5, 6\}$ and a relation R on A be defined by $xRy = x + y > 8$. Draw the digraph representation for R and find the matrix representation of R.

Solution:

$$R = \{(4,5), (4,6), (5,4), (5,5), (5,6), (6,4), (6,5), (6,6)\}.$$

The digraph representation of R is



The matrix representation/adjacency matrix is

$$M_R = \begin{array}{c} \\ 1 \\ 2 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

Definition 4.2.14. Given a digraph $G = (V, E)$. For a node $v \in V$, the ***in-degree of*** v, denoted as $\deg^-(v)$, is the number of head endpoints adjacent to the node v and the ***out-degree of*** v, denoted as $\deg^+(v)$, is the number of tail endpoints to v.

A node with $\deg^-(v) = 0$ is called a ***source*** whereas a node with $\deg^+(v) = 0$ is called a ***sink***.

Definition 4.2.15. A ***weighted digraph*** or ***network*** or ***labelled digraph*** is a digraph in which the edges are attached with a ***weight*** or ***label***. ***Weights*** are usually real numbers while labels can be real numbers or strings. The ***(weighted) adjacency matrix*** for a weighted digraph is a matrix with the weights of edges.

Weighted Digraph in Standard ML (`warshall_dig.sml`)

```
(* Digraph: Representing vertices with positive integers; 0 for empty graph *)
signature NODE =
sig
  exception NodeError
  eqtype node
  val nilNode : node
  val isNil : node -> bool
  val max : node * node -> node
  val toInt : node -> int
```

```
  val fromInt : int -> node
  val toString : node -> string
  val directedListToString : node list -> string
end

structure Node : NODE =
struct
  exception NodeError
  type node = int
  val nilNode = 0

  fun isNil u = u = nilNode
  fun max (u, v) = Int.max (u, v)
  fun toInt u = u

  fun fromInt i = if i < nilNode then raise NodeError else i

  fun toString u = Int.toString u

  fun directedListToString [] = ""
    | directedListToString [u] = toString u
    | directedListToString (u :: tail) =
      (* This implementation is *not* tail recursive. *)
      (toString u) ^ " -> " ^ (directedListToString tail)
end

(* Graph edges, with weights. *)

signature EDGE =
sig
  type edge
  val new : Node.node * real * Node.node -> edge
  val first : edge -> Node.node
  val weight : edge -> real
  val second : edge -> Node.node
  val print_dot : edge -> unit
end

structure Edge : EDGE =
struct
  type edge = Node.node * real * Node.node

  fun new edge = edge
  fun first (u, _, _) = u
  fun weight (_, w, _) = w
  fun second (_, _, v) = v
  fun print_dot (u, _, v) =
    print (Node.toString u ^ " -> " ^ Node.toString v ^ ";\n")
end

val R = [Edge.new (Node.fromInt 1, 1.0, Node.fromInt 2),
         Edge.new (Node.fromInt 2, 1.0, Node.fromInt 1),
         Edge.new (Node.fromInt 2, 1.0, Node.fromInt 3),
         Edge.new (Node.fromInt 3, 1.0, Node.fromInt 4)];
val _ = map Edge.print_dot R;
```

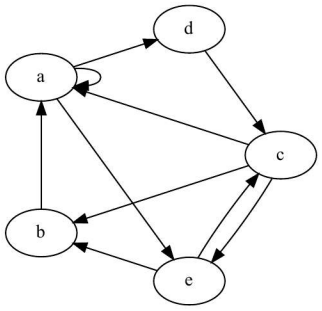
Example 4.2.16. Given $A = \{a, b, c, d, e\}$ and a matrix representation of R below:

$$M_R = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Draw the digraph representation for R and find the in-degree and out-degree of each node of the digraph representation.

Solution:

The digraph representation is



Node	a	b	c	d	e
In-degree	3	2	2	1	2
Out-degree	3	1	3	1	2

Example 4.2.17. Consider a digraph $G = (V, E)$ where $V = \{1, 2, 3, 4, 5\}$ and

$$E = \{(1, 2), (1, 4), (2, 1), (2, 3), (2, 4), (2, 5), (3, 5), (4, 5), (5, 1), (5, 5)\}.$$

Find $Adj(G)$ and the in-degree and out-degree of each node of G.

An important concept in the study of digraph is the path, which is defined as below. The concept of path is used to characterise the concept of "reachability", which is important in many areas of computer science. For example, a program using pointer-based data structures is free of memory leaks if every allocated segment of memory can be reached from a program variable. Finding solutions to solitaire puzzles, or more generally, goal search and motion planning can be expressed in terms of reachability.

Definition 4.2.18. For a digraph $G = (V, E)$, a ***path from a node*** a ***to a node*** b is a sequence of nodes

$$a, v_1, v_2, \cdots, v_{n-1}, b$$

such that $(a, v_1), (v_1, v_2), (v_2, v_3), \cdots, (v_{n-1}, b) \in E$. The ***length of a path*** is the number of edges in that path. A ***cycle*** is a path begins and ends at the same node, i.e. $a = b$. A ***non-null path*** is a path in which the length is positive. For any two nodes $s, t \in V$, t is called ***reachable*** from s if there is a path from s to t.

Weighted Digraph in Standard ML (`warshall_pth.sml`)

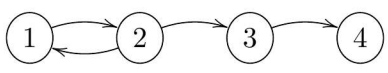
```
(* The "next" array and its operations. It lets you look up optimum paths. *)
signature PATHS =
sig
  type paths
  val new : int -> paths
  val get : paths -> int * int -> Node.node
  val set : paths -> int * int -> Node.node -> unit
  val path : (paths * int * int) -> Node.node list
  val pathString : (paths * int * int) -> string
end

structure Paths : PATHS =
struct
  type paths = Node.node SquareArray.squareArray

  fun new n = SquareArray.new (n, Node.nilNode)
  val get = SquareArray.get
  val set = SquareArray.set

  fun path (p, u, v) =
    if Node.isNil (get p (u, v)) then
      []
    else
      let
        fun build_path (p, u, v) =
        if u = v then
          [v]
        else
          u :: build_path (p, get p (u, v), v)
      in
        build_path (p, u, v)
      end

  fun pathString (p, u, v) =
    Node.directedListToString (path (p, u, v))
end
```

Given a digraph G with n nodes, how many paths of length k are there from one node to another? $[Adj(G)^2]_{ij}$ is the number of length 2 paths from node i to node j! Continuing in this fashion, the answer we desire can be found by computing $Adj(G)^k$!

Example 4.2.19. Consider a set $A = \{1, 2, 3, 4\}$ and its relation $R = \{(1, 2), (2, 1), (2, 3), (3, 4)\}$.

Write down all the paths of length 1 to length 4.

Solution: The digraph representation $G = (A, R)$ of R is



The adjacency matrix $Adj(G)$ of the digraph representation of R,

$Adj(G) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$ gives all length 1 paths: (1, 2), (2, 1), (2, 3), (3, 4)

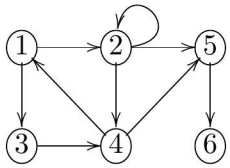
$Adj(G)^2 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$ gives all length 2 paths: (1, 2, 1), (1, 2, 3), (2, 1, 2), (2, 3, 4)

$Adj(G)^3 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$ gives all length 3 paths: (1, 2, 1, 2), (1, 2, 3, 4), (2, 1, 2, 1), (2, 1, 2, 3)

$Adj(G)^4 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$ gives all length 4 paths:

(1, 2, 1, 2, 1), (1, 2, 1, 2, 3), (2, 1, 2, 1, 2), (2, 1, 2, 3, 4)

Example 4.2.20. Consider the following digraph



1. Find path(s) from the node u to the node v with given length.

   (a) $u = 1$, $v = 4$, length=2. ..... 1, 2, 4 and 1, 3, 4

   (b) $u = 2$, $v = 6$, length=3. ..... 2, 2, 5, 6 and 2, 4, 5, 6

   (c) $u = 5$, $v = 3$, length=2. ..... No path from 5 to 3

2. Find all cycles with length 3.

Digraph theory and algorithms are used to solve practical problems related to connected nodes/states:

- *Transitive closure*: Is there a directed path from v to w?
- *Strong connectivity*: Are all nodes mutually reachable?
- *Topological sort*: Can another digraph be drawn so that all edges point from left to right?
- *PERT/CPM*[1]: Given a set of tasks with precedence constraints, how we can we best complete them all?
- *Shortest path*: Find best route from s to t in a weighted digraph.
- *PageRank*: What is the importance of a web page?

[1]Project Evaluation Review Technique/Critical Path Method

§4.3 Closures of (Binary) Relations

For a set $A = \{1, 2, 3\}$, if we are considering the addition relation on A, we find that $1 + 3$ gives 4, which not inside A.

This kind of relation (the addition on A) is said to be **not closed**.

By adding enough elements into A, we can form $\overline{A} = \{1, 2, 3, 4, \cdots\}$ which is closed for addition relation. Any two numbers adding together is still in A.

With the same rationale, we wish to turn some properties (e.g. reflexive, symmetry, etc.) of a binary relation to become "closed", i.e. all necessary elements to meet the property to be in R.

In the theory of rewriting systems (`https://en.wikipedia.org/wiki/Rewriting`), one often uses more wordy notions such as the reflexive transitive closure R^* — the smallest preorder containing R, or the reflexive transitive symmetric closure $R^{\equiv}$ — the smallest equivalence relation containing R, and therefore also known as the **equivalence closure**.

When considering a particular term algebra (`https://en.wikipedia.org/wiki/Term_algebra`), an equivalence relation that is compatible with all operations of the algebra is called a *congruence relation*. The **congruence closure** of R is defined as the smallest congruence relation containing R.

For arbitrary property P and relation R, the P **closure of** R **need not exist**.

The properties reflexivity, transitivity and symmetry are **closed under arbitrary intersections**. In such cases, the P closure can be directly defined as the intersection of all sets with property P containing the relation R.

Some important particular closures can be constructively obtained as follows:

- $cl_{ref}(R) = R \cup \{(x, x) : x \in S\}$ is the **reflexive closure** of R,
- $cl_{sym}(R) = R \cup \{(y, x) : (x, y) \in R\}$ is its **symmetric closure**,
- $cl_{trn}(R) = R \cup \{(x_1, x_n) : n > 1 \wedge (x_1, x_2), \ldots, (x_{n-1}, x_n) \in R\} = \bigcup_{i=1}^{n} R^i$ is its **transitive closure**,
- $cl_{emb,\Sigma}(R) = R \cup \{\langle f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n), f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)\rangle : \langle x_i, y\rangle \in R \wedge f \in \Sigma n\text{-ary} \wedge 1 \leq i \leq n \wedge x_1, \ldots, x_n \in S\}$ is its **embedding closure** with respect to a given set Σ of operations on S, each with a fixed arity.

The relation R is said to **have closure under some** cl_{xxx}, if $R = cl_{xxx}(R)$; for example R is called symmetric if $R = cl_{sym}(R)$.

In the latter case, the nesting order does matter; e.g. if S is the set of terms over $\Sigma = \{a, b, c, f\}$ and $R = \{\langle a, b\rangle, \langle f(b), c\rangle\}$, then the pair $\langle f(a), c\rangle$ is contained in the congruence closure of R, i.e. $cl_{trn}(cl_{emb,\Sigma}(cl_{sym}(cl_{ref}(R))))$, but not in the relation $cl_{emb,\Sigma}(cl_{trn}(cl_{sym}(cl_{ref}(R))))$.

Any of the four closures preserves symmetry, i.e., if R is symmetric, so is any $cl_{xxx}(R)$. Similarly, all four preserve reflexivity. Moreover, cl_{trn} preserves closure under $cl_{emb,\Sigma}$ for arbitrary Σ. As a consequence, the equivalence closure of an arbitrary binary relation R can be obtained as

$$cl_{trn}(cl_{sym}(cl_{ref}(R))),$$

and the congruence closure with respect to some Σ can be obtained as

$$cl_{trn}(cl_{emb,\Sigma}(cl_{sym}(cl_{ref}(R)))).$$

§4.3.1 Reflexive Closure

The reflexive closure of a relation R on a set A is the smallest set $cl_{ref}(R)$ that contains R and is reflexive.

Example 4.3.1. Consider a set $A = \{1, 2, 3, 4\}$ and its relation $R = \{(1, 2), (2, 1), (2, 3), (3, 4)\}$. Find the reflexive closure $cl_{ref}(R)$ of R.

Solution: $M_R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \Rightarrow M_{cl_{ref}(R)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$

We have $cl_{ref}(R) = \{\underline{(1,1),(2,2),(3,3),(4,4),(5,5),}(1,2),(2,1),(2,3),(3,4)\}$

§4.3.2 Symmetric Closure

A binary relation R may not be symmetric, but R is always contained in some symmetric relation $cl_{sym}(R)$. This is because we can add symmetric elements into R until we get the **smallest** relation R^{sym} is a symmetric closure.

Example 4.3.2. Consider a set $A = \{1, 2, 3, 4\}$ and its relation $R = \{(1, 2), (2, 1), (2, 3), (3, 4)\}$. Find the symmetric closure $cl_{sym}(R)$ of R.

Solution: $M_R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \Rightarrow M_{cl_{sym}(R)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$

We have $cl_{sym}(R) = \{(1, 2), (2, 1), (2, 3), (3, 4), \underline{(3, 2), (4, 3)}\}$

§4.3.3 Transitive Closure and Warshall's Algorithm

A binary relation R may not be transitive but it is always contained in some transitive relation $cl_{trn}(R)$, which is the smallest called the **transitive closure**.

The **intersection** of two transitive relations is transitive. However, the **union** of two transitive relations need not be transitive. To preserve transitivity, one must take the transitive closure. This occurs, for example, when taking the union of two equivalence relations or two preorders. To obtain a new equivalence relation (Definition 4.4.1) or a preorder relation (Definition 4.5.1) one must take the transitive closure (reflexivity and symmetry — in the case of equivalence relations — are automatic).

In logic, the transitive closure of a binary relation cannot solely be expressed in first-order logic (FO). This means that one cannot write a formula using predicate symbols R and T that will be satisfied in any model iff T is the transitive closure of R. In finite model theory, first-order logic (FO) extended with a transitive closure operator is usually called transitive closure logic, and abbreviated FO(TC) or just TC. TC is a sub-type of fixpoint logic. The fact that FO(TC) is strictly more expressive than FO was discovered by Ronald Fagin in 1974; the result was then rediscovered by Alfred Aho and Jeffrey Ullman in 1979, who proposed to use fixpoint logic as a database query language.

Finding the transitive closure of a digraph is an practical problem in many computational tasks. It is used, for instance, in the reachability analysis of transition networks representing distributed and parallel systems and in the construction of parsing automata in compiler construction. Recently, efficient transitive closure computation has been recognised as a significant sub-problem in evaluating recursive database queries, since almost all practical recursive queries are transitive.

Let M be the matrix representation of the binary relation R of a finite set A of n elements.
Naïve Algorithm for getting transitive closure:

The following matrix
$$M^+ = M \vee M^2 \vee \cdots \vee M^n.$$
gives the matrix representation of the transitive closure of R.

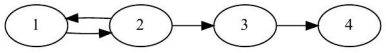
The naïve algorithm is very computationally inefficient! requiring n matrix multiplications!

(Floyd-)Warshall's Algorithm for getting transitive closure:

```
C := copy(M)
for k = 1, ··· , n
    for i = 1, ··· , n
        if i ≠ k and C_ik ≠ 0:
            for j = 1, ··· , n
                C_ij := C_ij ∨ C_kj
```

Example 4.3.3. Consider a set $A = \{1, 2, 3, 4\}$ and its relation $R = \{(1, 2), (2, 1), (2, 3), (3, 4)\}$ from Example 4.2.19. Find the transitive closure $cl_{trn}(R)$ of R.

Solution: The digraph representation of R is



Since $(1, 2) \in R$ and $(2, 1) \in R$ but $(1, 1) \notin R$, so R is not transitive.

The matrix representation of R is

$$M_R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Step $k = 1$: look for 1 in first column.

$$M_R^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & \boxed{1} & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Step $k = 2$: look for 1 in second column.

$$M_R^{(2)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} \boxed{1} & 1 & \boxed{1} & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Step $k = 3$: look for 1 in third column.

$$M_R^{(3)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & \boxed{1} \\ 1 & 1 & 1 & \boxed{1} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Step $k = 4$: look for 1 in forth column.

$$M_R^{(4)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

The transitive closure of R is $cl_{trn}(R) = \{(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (2, 4), (3, 4)\}$.

Warshall's Algorithm in Standard ML

```
(* https://rosettacode.org/wiki/Floyd-Warshall_algorithm *)
use "warshall_mat";   (* Boolean matrix representation *)
use "warshall_dig";   (* digraph representation *)
use "warshall_pth";   (* paths *)

(*---------------------------------------------------------------------*)

signature DISTANCES =
sig
  type distances
  val new : int -> distances
  val get : distances -> int * int -> real
  val set : distances -> int * int -> real -> unit
end

structure Distances : DISTANCES =
struct
  type distances = real SquareArray.squareArray
  fun new n = SquareArray.new (n, Real.posInf)
  val get = SquareArray.get
  val set = SquareArray.set
end

(*---------------------------------------------------------------------*)

exception FloydWarshallError

fun find_max_node [] = Node.nilNode
  | find_max_node (edge :: tail) =
    (* This implementation is *not* tail recursive. *)
    Node.max (Node.max (Edge.first edge, Edge.second edge),
              find_max_node tail)

(* Floyd-Warshall Algorithm: This implementation is unable to handle loop *)
fun floyd_warshall [] = raise FloydWarshallError
  | floyd_warshall edges =
    let
      val n = find_max_node edges
      val dist = Distances.new n
      val next = Paths.new n

      fun read_edges [] = ()
        | read_edges (edge :: tail) =
          let
            val u = Edge.first edge
            val v = Edge.second edge
            val weight = Edge.weight edge
          in
            (Distances.set dist (u, v) weight;
             Paths.set next (u, v) v;
             read_edges tail)
          end

      (* Indices in order from 1 .. n. *)
      val indices = List.tabulate (n, fn i => i + 1)
```

```
    in
      (* Initialization. *)
      read_edges edges;
      List.app (fn i => (Distances.set dist (i, i) 0.0;
                         Paths.set next (i, i) i))
               indices;

      (* Perform Floyd-Warshall computation *)
      List.app
        (fn k =>
            List.app
              (fn i =>
                  List.app
                    (fn j =>
                        let
                          val dist_ij = Distances.get dist (i, j)
                          val dist_ik = Distances.get dist (i, k)
                          val dist_kj = Distances.get dist (k, j)
                          val dist_ikj = dist_ik + dist_kj
                        in
                          if dist_ikj < dist_ij then
                            let
                              val new_dist = dist_ikj
                              val new_next = Paths.get next (i, k)
                            in
                              Distances.set dist (i, j) new_dist;
                              Paths.set next (i, j) new_next
                            end
                          else
                            ()
                        end)
                    indices)
              indices)
        indices;

      (* Return the results, as a 3-tuple. *)
      (n, dist, next)
    end

(*---------------------------------------------------------------*)

fun tilde_to_minus s =
    String.translate (fn c => if c = #"~" then "-" else str c) s

fun main () =
    let
      val (n, dist, next) = floyd_warshall R
      val indices = List.tabulate (n, fn i => i + 1)
    in
      print "  pair     distance    path\n";
      print "-------------------------------------\n";
      List.app
        (fn u =>
            List.app
              (fn v =>
                  if u <> v then
```

```
                    (print " ";
                     print (Node.directedListToString [u, v]);
                     print "     ";
                     if 0.0 <= Distances.get dist (u, v) then
                       print " "
                     else
                       ();
                     print (tilde_to_minus
                              (Real.fmt (StringCvt.FIX (SOME 1))
                                        (Distances.get dist (u, v))));
                     print ("      " ^ Paths.pathString (next, u, v) ^ "\n"))
                  else
                    ())
              indices)
        indices
    end;

main ();
```

Example 4.3.4. Consider a relation $R = \{(1,1),(1,2),(2,1),(2,2),(3,3),(3,4),(4,3),(4,4),(4,5),(5,4),(5,5)\}$ on $A = \{1,2,3,4,5\}$. Find the transitive closure of R.

§4.4 Equivalence Relation

Equivalence relation is the **generalisation** of the equality of numbers "=" to sets with the following properties:

- $x = x$ for all x
- if $x = y$, then $y = x$
- if $x = y$ and $y = z$, then $x = z$.

Definition 4.4.1. A binary relation R on a set A is called

- **reflexive**: xRx for all $x \in A$ (Definition 4.1.17);
- **symmetric**: if xRy, then yRx for all x, y (Definition 4.1.19);
- **transitive**: whenever $(x, y) \in R$ and $(y, z) \in R$, then $(x, z) \in R$ (Definition 4.1.22);
- a ***compatibility relation*** (`https://math.stackexchange.com/questions/508167/how-to-define-a-compatible-relation-using-a-cover`) or a ***tolerance relation*** (`https://en.wikipedia.org/wiki/Tolerance_relation`) if it is reflexive and symmetric;
- a ***equivalence relation*** if it a compatibility relation which is transitive.

Example 4.4.2. Is the inequality relation "$<$" an equivalence relation on $\mathbb{Z}$?

Solution: No, since the relation $<$ is not a reflexive (e.g. $1 \not< 1$).

Example 4.4.3. Let $A = \mathbb{Z}$ and R be the relation on A defined by $aRb = a + b$ is even. Show that R is an equivalence relation.

Proof:

- $a + a = 2a$ is even, so $(a, a) \in R$. R is reflexive.
- If $a + b$ is even then $b + a$ is also even. Thus $(a, b) \in R \Rightarrow (b, a) \in R$. So R is symmetric.
- Let $a, b, c \in A$. Let $(a, b) \in R$ and $(b, c) \in R$, then $a + b = 2k_1$ and $b + c = 2k_2$ for some integer k_1 and k_2. Then $a + c = 2k_1 + 2k_2 - 2b = 2(k_1 + k_2 - b)$ is even. So R is transitive.

Hence R is an equivalence relation.

Definition 4.4.4. Let R be an equivalence relation on a set A and let $a \in A$. The set of all elements that related to a, is called the ***equivalence class of*** a and is denoted as

$$[a] \text{ or } \overline{a} = \{x \in A \ : \ (x, a) \in R\}.$$

The set of all the distinct equivalence classes of R form a set A/R, called the ***quotient set of*** A.

Example 4.4.5 (Finite Set)**.** Consider the set

$$A = \{a, b, c, d, e, f, g\}$$

and the equivalence relation

$$R = \{(a, a), (a, b), (b, a), (b, b), (c, c), (c, d), (d, c), (d, d), (e, e), (e, f), (e, g), (f, e), (f, f), (f, g), (g, e), (g, f), (g, g)\}.$$

Find the equivalence class of each element in A and the quotient set A/R.

Solution: The equivalence class of

- a, b is $\{a, b\}$
- c, d is $\{c, d\}$
- e, f, g is $\{e, f, g\}$

The quotient set $A/R = \{\{a, b\}, \{c, d\}, \{e, f, g\}\}$

Example 4.4.6. Let $A = \{1, 2, 3, 4\}$ and $R = \{(1, 1), (1, 2), (2, 1), (2, 2), (3, 3), (4, 4)\}$ be an equivalence relation on A. Find the equivalence class of each element in A and the quotient set A/R.

Example 4.4.7. Let $A = \mathbb{Z} = \{0, \pm 1, \pm 2, \dots\}$ and R be defined by $aRb = a + b$ is even. Find the equivalence class of each element in A and the quotient set A/R.

Solution: We put in some values to learn R concretely:

$$0R0, 0R2, 0R4, ..., 0R(-2), 0R(-4), ...$$
$$1R1, 1R3, 1R5, ..., 1R(-1), 1R(-3), 1R(-5), ...$$

Unfolding: It just means

0+0 is even, 0+2 is even, 0+4 is even, ... etc.

There are two equivalence classes:

- $[0] = \{0, \pm 2, \pm 4, \dots\}$
- $[1] = \{\pm 1, \pm 3, \pm 5, \dots\}$

The quotient is

$$A/R = \{[0],\ [1]\} = \{\{0, \pm 2, \pm 4, \dots\},\ \{\pm 1, \pm 3, \pm 5, \dots\}\}.$$

Example 4.4.8. Let $A = \mathbb{Z}^+$ and R be the relation on A defined by

$$R = \{(x, y) \in A \times A \ :\ x - y \mod 3 = 0\}.$$

Find the equivalence class (called the "modulus" or "modular numbers") of each element in A and the quotient set of A.

Solution: Since

- for all $x, y \in \{1, 4, 7, 10, \cdots\}$, $x - y \mod 3 = 0$;
- for all $x, y \in \{2, 5, 8, 11, \cdots\}$, $x - y \mod 3 = 0$;
- for all $x, y \in \{3, 6, 9, 12, \cdots\}$, $x - y \mod 3 = 0$;

we have

$$[1] = \{1, 4, 7, 10, \cdots\} = [4] = [7] = \cdots = [3k - 2]$$
$$[2] = \{2, 5, 8, 11, \cdots\} = [5] = [8] = \cdots = [3k - 1]$$
$$[3] = \{3, 6, 9, 12, \cdots\} = [6] = [9] = \cdots = [3k]$$

where $k = 1, 2, \cdots$.

So there are only three distinct equivalent classes of R, and hence $A/R = \{[1], [2], [3]\}$.

Remark 4.4.9. Example 4.4.8 is just a special case of the fact that the congruence modulo n is an equivalence relation.

- reflexive: For any $x \in \mathbb{Z}$, $x \equiv x \pmod{n}$;
- symmetric: For any $x, y \in \mathbb{Z}$, if $x \equiv y \pmod{n}$, then $y \equiv x \pmod{n}$;
- transitive: For any $x, y, z \in \mathbb{Z}$, if $x \equiv y \pmod{n}$ and $y \equiv z \pmod{n}$, then $x \equiv z \pmod{n}$.

The quotient $\mathbb{Z}/n\mathbb{Z} = \mathbb{Z}/\equiv_n$ is $\{[0], [1], \cdots, [n-1]\}$
where

$$\begin{aligned}
[0] &= \{0, \pm n, \pm 2n, \cdots\} = n\mathbb{Z} \\
[1] &= \{1, \pm n + 1, \pm 2n + 1, \cdots\} = n\mathbb{Z} + 1 \\
&\cdots \\
[n-1] &= \{n-1, \pm n + (n-1), \pm 2n + (n-1), \cdots\} = n\mathbb{Z} + (n-1)
\end{aligned}$$

Mathematicians construct new mathematical objects from know mathematical objects using **(set) product**, **subset**, **(set) union** and **(set) quotient** (based on some kind of equivalence relations).

Construction of the Set of Integers $\mathbb{Z}$

Assume we have a natural number set $\mathbb{N}$. The integer set $\mathbb{Z}$ is defined as the quotient set $(\mathbb{N} \times \mathbb{N})/\sim$ with

$$(m_1, n_1) \sim (m_2, n_2) \equiv m_1 + n_2 = m_2 + n_1$$

where $m_1, m_2, n_1, n_2 \in \mathbb{N}$. So $0 = \{(i, i) : i \in \mathbb{N}\}$, $1 = \{(i+1, i) : i \in \mathbb{N}\}$, $-1 = \{(i, i+1) : i \in \mathbb{N}\}$, etc.

Construction of the Set of Rational Numbers $\mathbb{Q}$

$\mathbb{Q} = \mathbb{Z} \times \mathbb{Z}/\sim$ where the equivalence relation $\sim$ is given by

$$(p_1, q_1) \sim (p_2, q_2) \equiv p_1 q_2 = p_2 q_1.$$

So we have $0 = \{(0, i) : i \in \mathbb{Z}\}$, $\frac{1}{2} = \{(i, 2i) : i \in \mathbb{Z}\}$, etc.

Construction of the Set of Real Numbers $\mathbb{R}$

There are many methods to construct $\mathbb{R}$ from a lot of $\mathbb{Q}$ which are introduced in `https://en.wikipedia.org/wiki/Construction_of_the_real_numbers`

1. Equivalent Cauchy sequence
2. Dedekind cuts
3. ... less popular.

Construction of the Euclidean Space $\mathbb{R}^n$ (Linear Algebra)

By using set product, we can obtain the Euclidean Space

$$\mathbb{R}^n = \mathbb{R} \times \mathbb{R} \times \cdots \times \mathbb{R}.$$

However, we need to define the necessary operations such as equality, addition, subtraction, scalar multiplication, dot product, etc.

Construction of the Geometrical Objects

We construct geometrical objects (e.g. a circle) using subset in a product space like the Euclidean space or using equivalence relations on various sets to **glue** them together like a sculptor (these mathematicians are called topologists).

Example 4.4.10. $S^1 = \{(x, y) \in \mathbb{R}^2 \ : \ x^2 + y^2 - 1 = 0\} \subset \mathbb{R}^2$

Example 4.4.11. $S^n = \{(x_1, ..., x_n) \in \mathbb{R}^n \ : \ x_1^2 + \cdots, x_n^2 - 1 = 0, \ j = 1, ..., m\}$.

If X is a topological space (Definition 4.5.50), gluing the points x and y in X means considering the quotient space obtained from the equivalence relation

$$a \sim b \text{ iff } a = b \text{ or } a = x, \ b = y.$$

For example, consider $[0, 1]$ and the equivalence relation $\sim$ such that $0 \sim 1$ while $x \sim x$ for $0 < x < 1$, then $[0, 1]/\sim$ is homeomorphic to the circle S^1.

Consider the unit square $I^2 = [0, 1] \times [0, 1]$ and the equivalence relation $\sim$ generated by the requirement that all boundary points be equivalent, thus identifying all boundary points to a single equivalence class. Then $I^2/\sim$ is homeomorphic to the sphere S^2.

More generally, suppose X is a space and A is a subspace of X. One can identify all points in A to a single equivalence class and leave points outside of A equivalent only to themselves. The resulting quotient space is denoted X/A (as an abbreviation of X/R).

Consider the set $\mathbb{R}$ of real numbers with the ordinary topology, and write $x \sim y$ iff $x - y \in \mathbb{Z}$. Then the quotient space $X/\sim$ is homeomorphic to the unit circle S^1 via the homeomorphism which sends the equivalence class of x to $\exp(2\pi ix)$.

A generalisation of the previous example is the following: Suppose a topological group G acts continuously on a space X. One can form an equivalence relation on X by saying points are equivalent iff they lie in the same orbit. The quotient space under this relation is called the **orbit space**, denoted X/G.

There are YouTube videos on the application of equivalence relation to construct interesting geometrical objects:

- `https://www.youtube.com/watch?v=lmcT2mP2bfE`

Most of them are boring mathematics because it is difficult to create pretty video explaining how to use product, quotient, etc. to create new mathematical objects called manifolds.

Many equivalence relations which are just "impossible" to visualise.

Example 4.4.12 (`https://en.wikipedia.org/wiki/Projective_space`)**.** A **projective space** $\mathbb{P}(\mathbb{R}^n)$ of dimension n is the quotient set of $\mathbb{R}^{n+1} \setminus \{0\}$ by the equivalence relation "being on the same vector line", i.e. $x \sim y$ if there is a nonzero element $\lambda \in \mathbb{R}$ such that $x = \lambda y$.

A generalisation of projective space is the Grassmannian $Gr(k, \mathbb{R}^n)$ which is a set that parameterises all k-dimensional linear subspaces of the n-dimensional vector space $\mathbb{R}^n$. When $k = 1$, $Gr(1, \mathbb{R}^n)$ is the projective space $\mathbb{P}(\mathbb{R}^{n-1})$

There are many equivalence relations where the quotient is **impossible** to describe such as the logical equivalence relation: Let ϕ, ψ, ξ be any propositions.

- $\phi \leftrightarrow \phi$, therefore $\phi \equiv \phi$ and $\equiv$ is reflexive;
- If $\phi \equiv \psi$, then $\phi \leftrightarrow \psi$ is a tautology, then $\phi \leftrightarrow \psi$ is a tautology, therefore $\psi \equiv \phi$ and $\equiv$ is symmetric.
- If $\phi \equiv \psi$ and $\psi \equiv \xi$, then $\phi \to \psi$, $\phi \leftarrow \psi$, $\psi \to \xi$, $\psi \leftarrow \xi$ are tautologies, then $\phi \to \xi$ and $\phi \leftarrow \xi$ by modus ponens and $\phi \leftrightarrow \xi$ is a tautology, therefore $\phi \equiv \xi$ and $\equiv$ is transitive.

However, the quotient of all propositions under logical equivalence is **too complex** to describe.

§4.5 Partial Order Relation

Partial order relation is the **generalisation** of the ordering "$\leq$" to sets with the following properties:

- $x \leq x$
- if $x \leq y$ and $y \leq x$, then $x = y$
- if $x \leq y$ and $y \leq z$, then $x \leq z$.

for all x, y, z. The investigation of partial order relations has lead to the development of lattice theory in mathematical science but computer scientists have extended the theory to domain theory, which can be applied to denotational semantics of (functional) programming languages.

Definition 4.5.1. A binary relation R on a set A is called

- **reflexive**: xRx for all $x \in A$ (Definition 4.1.17);
- **antisymmetric**: if xRy and yRx, then $x = y$ for all $x, y \in A$ (Definition 4.1.19);
- **transitive**: whenever $(x, y) \in R$ and $(y, z) \in R$, then $(x, z) \in R$ (Definition 4.1.22);
- ***preorder*** if it is reflexive and transitive (`https://en.wikipedia.org/wiki/Preorder`);
- ***(partial) order*** or **approximation ordering** if it is reflexive, antisymmetric and transitive (`https://en.wikipedia.org/wiki/Partially_ordered_set`).

The set A together with the partial order relation $\preceq$, $(A, \preceq)$, is called a ***partially ordered set***, or ***poset***. For $x, y \in A$, y **approximates** x if $x \preceq y$.

Remark 4.5.2. Computer scientists use $\sqsubseteq$ rather than $\preceq$.

Example 4.5.3. Typical examples of partial order relation and posets:

- $\subset$ and $(\mathscr{P}(S), \subset)$;
- $\leq$ and $(\{1, 2, 3\}, \leq)$;
- $\geq$ and $(\{1, 2, 3\}, \geq)$;
- $\leq$ and $(\mathbb{Z}, \leq)$;
- $\geq$ and $(\mathbb{Z}, \geq)$;
- $\leq$ and $(\mathbb{R}, \leq)$;
- $\geq$ and $(\mathbb{R}, \geq)$;
- (A, R) where $A =$ English dictionary, $xRy = x$ comes before y in dictionary order;
- (A, R) where $A =$ set of project tasks, xRy means that x must be completed before y begins where x and y are tasks in a project A.

Example 4.5.4. $(\mathbb{Z}, <)$ and $(\mathbb{Z}, >)$ are **not** posets since the relations "$<$" and "$>$" are not reflexive: $1 \not< 1$ and $1 \not> 1$.

Example 4.5.5. Show that $(\mathscr{P}(\{1, 2, 3\}), \subset)$, a special case of Example 4.5.3 is a poset.

Proof:

$$\mathscr{P}(\{1, 2, 3\}) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

The matrix representation

$$M_{\subset} = \begin{array}{c} \\ \emptyset \\ \{1\} \\ \{2\} \\ \{3\} \\ \{1,2\} \\ \{1,3\} \\ \{2,3\} \\ \{1,2,3\} \end{array} \begin{array}{c} \begin{array}{cccccccc} \emptyset & \{1\} & \{2\} & \{3\} & \{1,2\} & \{1,3\} & \{2,3\} & \{1,2,3\} \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

The $\subset$ is reflexive because the diagonal elements of $M_{\subset}$ are all 1.

The $\subset$ is antisymmetric because the lower triangular part of $M_{\subset}$ are all 0 so not symmetric pair can be formed.

The $\subset$ is transitive since if $A \subset B$ and $B \subset C$, then $A \subset C$.

Attempting to use M_R^2 to verify transitivity would be too painful:

$$M_{\subset}^2 = \begin{array}{c} \\ \emptyset \\ \{1\} \\ \{2\} \\ \{3\} \\ \{1,2\} \\ \{1,3\} \\ \{2,3\} \\ \{1,2,3\} \end{array} \begin{array}{c} \begin{array}{cccccccc} \emptyset & \{1\} & \{2\} & \{3\} & \{1,2\} & \{1,3\} & \{2,3\} & \{1,2,3\} \end{array} \\ \begin{pmatrix} 1 & 2 & 2 & 2 & 4 & 4 & 4 & 8 \\ 0 & 1 & 0 & 0 & 2 & 2 & 0 & 4 \\ 0 & 0 & 1 & 0 & 2 & 0 & 2 & 4 \\ 0 & 0 & 0 & 1 & 0 & 2 & 2 & 4 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Since every non-zero element in the matrix $M_{\subset}^2$ corresponds to a non-zero element in $M_{\subset}$, $\subset$ is transitive.

The "divisibility" in Topic 3 are reflexive, antisymmetric and transitive and is therefore a partial order relation.

Example 4.5.6. Show that $(\mathbb{Z}^+, |)$ a poset where $|$ is the 'integral divisibility'.

Proof:

- The relation $|$ is reflexive because any positive integer divides itself.
- Note that if $a|b$ and $b|a$ then $a = \pm b$. Since $a, b \in \mathbb{Z}^+$, the negative sign will not happen and $a = b$. So the relation $|$ is antisymmetric.
- The relation $|$ is transitive because $a|b \Rightarrow b = k_1 a$, $b|c \Rightarrow c = k_2 b$, therefore $c = k_2 k_1 a \Rightarrow a|c$.

Hence the relation of divisibility $|$ is a partial order and $(\mathbb{Z}^+, |)$ is a poset.

In the poset $(\mathbb{Z}^+, |)$

- $2|4$, therefore $2 \preceq 4$
- $2|6$, therefore $2 \preceq 6$
- $4 \nmid 6$, therefore $4 \not\preceq 6$

- $6 \nmid 4$, therefore $6 \not\preceq 4$

Note that **neither** $4 \preceq 6$ **nor** $6 \preceq 4$, we say that 4 and 6 are **incomparable** in $(\mathbb{Z}^+, \preceq)$.

Definition 4.5.7. In a poset $(A, \preceq)$, for any a, $b \in A$, if neither $a \preceq b$ nor $b \preceq a$, then a and b are said to be ***incomparable***.

Definition 4.5.8. We say $a \prec b$ if $a \preceq b$ and $a \neq b$. Similarly, $a \succ b$ if $a \succeq b$ and $a \neq b$.

Definition 4.5.9. Given a poset $(A, \preceq)$. For $a \in A$,

- If there is no $b \in A$ such that $b \prec a$, we say that a is a ***minimal element (of A)*** or ***least element (of A)***.
- If there is some $b \in A$ so that $b \prec a$ and there is no $c \in A$ so that $b \prec c \prec a$, we say that b is an ***immediate predecessor of*** a.
- If there is no $b \in A$ such that $a \prec b$, we say that a is a ***maximal element (of A)***.
- If there is some $b \in A$ so that $a \prec b$ and there is no $c \in A$ so that $a \prec c \prec b$, we say that b is an ***immediate successor of*** a.

Example 4.5.10. In $(\mathbb{Z}^+, |)$:

- 1 is the minimal element of $\mathbb{Z}^+$ because $1|a$, $a \in \mathbb{Z}^+$.
- 1 is the immediate predecessor of any prime number
- 2 is the immediate predecessor of 4, 6 but not 8 because $2 \prec 4 \prec 8$; 4 is the immediate successor of 2, 6 is the immediate successor of 2 and 3, etc.
- There is no maximal element in $\mathbb{Z}^+$

Example 4.5.11. $(\mathbb{Q}, \leq)$ has no minimal element, maximal element and any $a \in \mathbb{Q}$ does not have immediate predecessor or successor.

A special case of poset is chain.

Definition 4.5.12. A ***total order*** or ***linear order*** is a partial order relation on some set A which is **total**, i.e. for all $x, y \in A$:

- $x \preceq y$ or $y \preceq x$ (totality).

A set paired with a total order (A, R) is called a ***totally ordered set***, a ***linearly ordered set***, or a ***chain***.

Example 4.5.13. $\leq$ and $\geq$ are total order on $\mathbb{Z}$. $(\mathbb{Z}, \leq)$ and $(\mathbb{Z}, \geq)$ are chains.

A particular case of total order is the well-order.

Definition 4.5.14. A ***well-order*** is a total order such that

- every non-empty subset has a least element.

A set A with its well-order $\preceq$ forms a ***well-ordered set*** $(A, \preceq)$.

Example 4.5.15 (`https://en.wikipedia.org/wiki/Well-order`).

- $\leq$ on $\mathbb{R}$, $\mathbb{Z}$ is not well-order because the non-empty subset $\{-2, -4, -6, ...\}$ does not have a least element.
- $\leq$ on $\mathbb{N}$ is well order.

- The order $\preceq$ on $\mathbb{N}$ such that

$$0 \preceq 2 \preceq 4 \preceq 6 \preceq 8 \preceq ... \preceq 1 \preceq 3 \preceq 5 \preceq 7 \preceq ...$$

  is a well order.

- The order $\preceq$ on $\mathbb{Z}$ such that

$$0 \preceq 1 \preceq 2 \preceq 3 \preceq 4 \preceq ... \preceq -1 \preceq -2 \preceq -3 \preceq -4 \preceq ...$$

  is a well order.

- The order $\preceq$ on $\mathbb{Z}$ such that

$$0 \preceq -1 \preceq 1 \preceq -2 \preceq 2 \preceq -3 \preceq 3 \preceq -4 \preceq 4 \preceq ...$$

  is a well order.

Theorem 4.5.16 (Well-Ordering Principle)**.** *For any non-empty set A, there is a well-order $\prec$.*

Partial orders form a natural setting for monotonic functions.

Definition 4.5.17. Given two posets $(A, \preceq_A)$ and $(B, \preceq_B)$, a function, $f : A \to B$, is

- **monotonic**, **increasing** or **order-preserving** if for all $a, b \in A$, $a \preceq_A b \Rightarrow f(a) \preceq_B f(b)$.
  When $\preceq$ is changed to $\prec$, we call f **strictly increasing**.
- **decreasing** if for all $a, b \in A$, $a \preceq_A b \Rightarrow f(b) \preceq_B f(a)$.
  When $\preceq$ is changed to $\prec$, we call f **strictly decreasing**.

The following is a generalisation of mathematical induction.

Theorem 4.5.18 (Principle of Transfinite Induction)**.** *Let $(A, \preceq)$ be a well-ordered set with a minimal element x_0.*

1. *Let $B \subset A$ and the initial segment of the set A with respect to the element z be $A_z := \{x \in A \ : \ x \prec z\}$. For any $z \in A$, when $A_z \subset B$, $z \in B$. Then $B = A$.*
2. *Suppose $P(x)$ is a predicate with $x \in A$ such that*
   (a) $P(x_0)$ is true;
   (b) For any $x \in A$, if $\forall y(y \prec x \to P(y))$, then $P(x)$ is true.

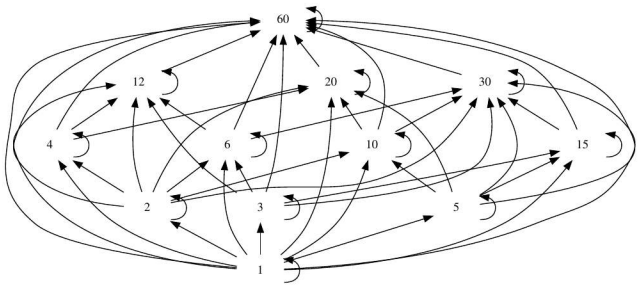
   Then $\forall x \in A,\ P(x)$.

Definition 4.5.19. A digraph (Definition 4.2.9) that contains no sequence of edges leading back to its starting point is called a **directed acyclic graph** or **DAG**.

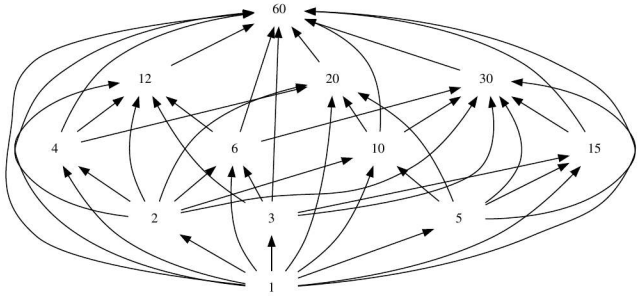
Definition 4.5.20. A **Hasse diagram** is a "reduced" DAG to represent the **transitive reduction**. For a finite poset $(S, \preceq)$, one represents each element of S as a vertex in the plane and draws a line segment or curve that goes upward from x to y whenever $x \preceq y$ and there is no z such that $x \preceq z \preceq y$. These curves may cross each other but must not touch any vertices other than their endpoints. Such a diagram, with labelled vertices, uniquely determines its partial order.

Let D_n denote the **set of all divisors of the integer** n.

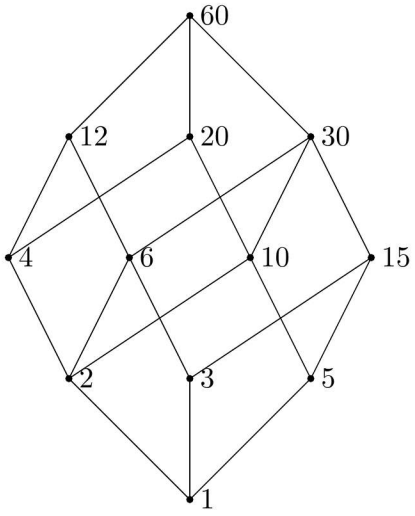
Example 4.5.21. Sketch the digraph representation, DAG and Hasse diagram of the poset $(D_{60}, |)$.



digraph of $(D_{60}, |)$

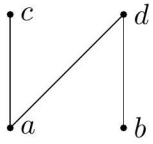
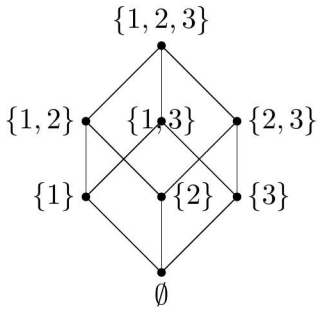


DAG of $(D_{60}, |)$



Hasse Diagram

Example 4.5.22.

$(\mathbb{Z}^{\geq 0}, \leq)$ and $(\mathbb{Z}^{\geq 0}, \geq)$ are posets since the relations "$\leq$" and "$\geq$" are reflexive, antisymmetric and transitive on $\mathbb{Z}$.	$(\{a, b, c, d\}, \preceq)$ where $a \preceq c$, $a \preceq d$, $b \preceq d$ and $e \preceq e$ for every $e = a, b, c, d$. 	$(\mathscr{P}(\{1,2,3\}), \subset)$ is a poset. 

Example 4.5.23. $(D_{12} = \{1, 2, 3, 4, 6, 12\}, |)$ is a poset. Let the elements of $|$ and sketch the digraph representation and the Hasse diagram.

It would be interesting to generalise the factors of an integer to "factors of polynomials".

The first difficulty we would face would be factors for a polynomial are not unique. For example,

$$x^2 + 5x + 6 = (x + 2)(x + 3) = 6(1 + \frac{x}{2})(1 + \frac{x}{3}).$$

To make the factorisation unique, we have to limit ourselves to polynomials with the coefficient of the higher degree being 1. In such a case, the factorisation will be the middle term and is unique.

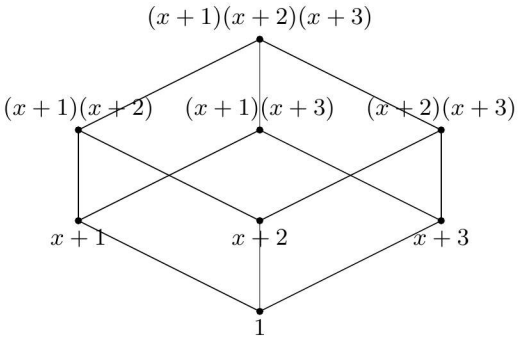
Definition 4.5.24. A ***monic polynomial*** is a nonzero univariate polynomial in with the leading coefficient is equal to 1, i.e. it has the form

$$x^n + c_{n-1}x^{n-1} + \cdots + c_2x^2 + c_1x + c_0$$

for some $n \geq 0$.

Example 4.5.25 (Monic Factors of a Monic Polynomial)**.** Let A be all the monic polynomial factors of the monic polynomial $(x + 1)(x + 2)(x + 3)\}$ and the partial order relation $p \preceq q$ be p divides q for any monic polynomials p, q. Sketch the Hasse diagram of the poset $(A, \preceq)$.

Solution:



It seems that we can obtain a Hasse diagram representing a distributive lattice isomorphic to the lattice given by the Boolean algebra generated by $(\mathscr{P}(\{1, 2, 3\}, \subset)$.

Remark 4.5.26. The lack of uniqueness in factorisation of algebraic objects such as the set of polynomials $\mathbb{R}[x]$ forces mathematicians to develop "abstract modern algebra" to characterise the components of algebraic objects. This is also why the notion of lattice in Section 4.5.1 is not popular in mathematics.

Example 4.5.27 (Abstraction from English dictionary)**.** Let Σ be some **alphabet** and consider the set

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$$

of all **finite words** drawn from Σ.

Given two words x and y, we define the partial order

$$x \preceq y \ := \ x \text{ is a prefix of } y,$$

i.e. if there is some word z such that $xz = y$.

$(\Sigma^*, \preceq)$ can be shown to be a poset.

Methods for constructing new poset from two posets $(A, \preceq_A)$ and $(B, \preceq_B)$ are given below.

Method 1

Define $\preceq$ on $A \times B$ to be

$$(a, b) \preceq (a', b') \ := \ a \preceq_A a' \text{ and } b \preceq_B b'.$$

It can be proved that $\preceq$ is a partial order. The poset $(A \times B, \preceq)$ defined in this way is called the **product poset** of A and B.

Method 2

Define $\preceq$ on $A \times B$ to be

$$(a, b) \preceq (a', b') \ := \ (a \preceq_A a') \vee (a = a' \wedge b \preceq_B b).$$

It is a partial order on $A \times B$ called the **lexicographic order**.

The useful property of lexicographic order (lex order for short) is that if the original partial orders are total, so is the lex order: this is why dictionary-makers use it. This also gives a source of very difficult-to-visualise total orders, like lex order on $\mathbb{R} \times \mathbb{R}$, which looks like the classic real number line where every point is replaced by an entire copy of the reals.

§4.5.1 Lattice

A lattice is a special kind of poset which has algebraic structures.

There are two notions of lattice in mathematics and computer science. This makes searching for "lattice" on the Internet a difficult task if one knows little about it.

One refers to the algebraic object with meet and join operations which is introduced in this section. They are used in distributed computing (vector clocks and global predicate detection), concurrency theory (pomsets and occurence nets), computability theory (recursion theory), programming language semantics (fixed-point semantics), etc. [Garg, 2015]

Another one refers to the geometric object with regular/periodic patterns in a high dimensional vector space and are used in cryptography and coding theory.

Definition 4.5.28. [Cartwright et al., 2016] Let $(A, \preceq)$ be a poset and $X \subset A$.

- An element $b \in A$ is an ***upper bound*** of X iff $x \preceq b$ for all $x \in X$.
- An element $x \in A$ is a ***lower bound*** of X iff $b \preceq x$ for all $x \in X$.
- X is ***consistent*** (or **bounded**) iff X has an upper bound.
- An upper bound b of X is the ***least upper bound*** of X (denoted $\bigvee X$ or $\bigsqcup X$ (computer science) or $lub(X)$) iff b approximates all upper bounds of X.
- A lower bound b of X is the ***greatest lower bound*** of X (denoted $\bigwedge X$ or $\bigsqcap X$ (computer science) or $glb(X)$) iff every lower bound of X approximates b.

Remark 4.5.29. Upper bounds are much more important in domain theory than lower bounds.

Definition 4.5.30. A ***join-semilattice*** is a poset $(L, \preceq)$ where every two elements $a, b \in L$ has a **join** (i.e. Definition 4.5.28's least upper bound), denoted by $a \vee b$.

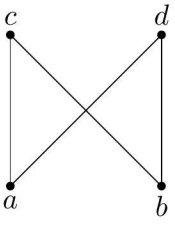
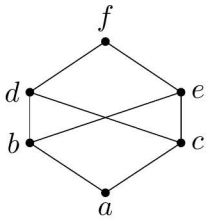
A ***meet-semilattice*** is a poset $(L, \preceq)$ where every two elements $a, b \in L$ has a **meet** (i.e. Definition 4.5.28's greatest lower bound), denoted by $a \wedge b$.

A poset $(L, \preceq)$ is called a **lattice** $(L, \wedge, \vee)$ if it is both a join-semilattice and a meet-semilattice, i.e. each two-element subset $\{a, b\} \subset L$ has a **join** and a **meet**.

Both join and meet are monotone with respect to the given order, i.e. when $a_1 \preceq a_2$ and $b_1 \preceq b_2$,

$$a_1 \vee b_1 \preceq a_2 \vee b_2 \quad \text{and} \quad a_1 \wedge b_1 \preceq a_2 \wedge b_2.$$

Example 4.5.31. Most posets are not lattices.

		
This poset is not a lattice because a and b do not have a common upper/lower bound.	This poset is not a lattice because c and d have no common upper bound.	This poset is not a lattice because b and c have common upper bounds d, e, f but none of them is the least upper bound.

Definition 4.5.32. A ***bounded lattice*** $((L, \vee, \wedge, 0, 1)$ is a lattice that has a greatest element (also called ***maximum***, or ***top element***, and denoted by 1 or by $\top$) and a least element (also called ***minimum***, or ***bottom***, denoted by 0 or by $\bot$), which satisfy

$$0 \preceq x \preceq 1, \quad x \vee 0 = x, \quad x \wedge 1 = x.$$

for every $x \in L$.

A poset is a bounded lattice if and only if every finite set of elements (including the empty set) has a join and a meet.

Definition 4.5.33. A poset is called a ***complete lattice*** if all its subsets have both a join and a meet.

Example 4.5.34.

1. For any set A, the poset $(\mathscr{P}(A), \subset)$ can be turned into a complete lattice $(\mathscr{P}(A), \cap, \cup, \emptyset, A)$.
2. The poset of all divisors of the integer $(D_n, |)$ can be turned into a complete lattice $(D_n, \gcd, \mathrm{lcm}, 1, n)$ where gcd stands for greatest common divisor (which is discussed in Topic 3) while lcm stands for least common multiplier.
3. The interval $([0, 1], \leq)$ can be turned into a complete lattice $([0, 1], \min, \max, 0, 1)$.

Remark 4.5.35. The interval $([0, 1] \cap \mathbb{Q}, \leq)$ can be turned into a bounded lattice $([0, 1], \min, \max, 0, 1)$. It is not a complete lattice because the set $\{r : r^2 \leq 1/2\}$ has no least upper bound.

Theorem 4.5.36. *Every complete lattice is a bounded lattice.*

Theorem 4.5.37. *If every subset of a poset L has a meet, then every subset of L has a join, hence L is a complete lattice.*

Definition 4.5.38. A ***continuous lattice*** $(L, \vee, \wedge, 0, 1)$ is a complete lattice which every element has a least upper bound (Definition 4.5.28):

$$x = \bigvee \{y \in L \ : \ x \in Int\{z \in L \ : \ y \preceq z\}\} \quad \text{for all } x \in L.$$

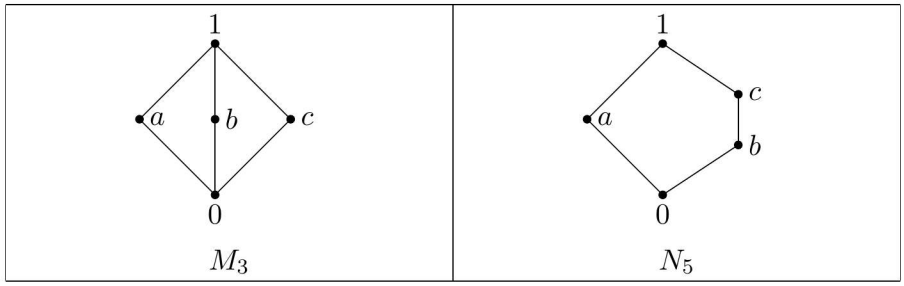
where the interior Int is taken in the sense of the induced topology [Scott, 1972].

Since lattices come with two binary operations, it is natural to ask whether one of them distributes over the other, that is, whether one or the other of the following distribution laws holds for every three elements $a, b, c \in L$:

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c); \quad a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c).$$

Definition 4.5.39. A lattice that satisfies the distribution laws is called a ***distributive lattice***.

The only non-distributive lattices with fewer than 6 elements are called M_3 and N_5, shown below.



Theorem 4.5.40. *A lattice is distributive iff it does not have a sublattice isomorphic to M_3 or N_5.*

Theorem 4.5.41. *Each distributive lattice is isomorphic to a lattice of sets i.e. $(\mathscr{P}(A), \cap, \cup, \emptyset, A)$ in Example 4.5.34.*

Definition 4.5.42. Let L be a bounded lattice $(L, \wedge, \vee, 0, 1)$. Two elements x and y of L are ***complements*** of each other if

$$x \vee y = 1 \quad \text{and} \quad x \wedge y = 0.$$

Some elements of a bounded lattice might not have a complement, and others might have more than one complement.

Example 4.5.43. $(L = \{0, 1/2, 1\}, \leq)$ is a bounded lattice, and $1/2$ does not have a complement because there is no $\ell \in L$ such that $\min\{\ell, 1/2\} = 0$ and $\max\{\ell, 1/2\} = 1$.

Example 4.5.44. N_5 is a bounded lattice such that a has two complements, viz. b and c.

Definition 4.5.45. A bounded lattice for which every element has a complement is called a ***complemented lattice*** (`https://en.wikipedia.org/wiki/Complemented_lattice`). A complemented lattice that is also distributive is a ***Boolean algebra*** $(B, \wedge, \vee, ', 0, 1)$ (`https://en.wikipedia.org/wiki/Boolean_algebra_%28structure%29`).

Example 4.5.46. Let A be a set. $(\mathscr{P}(A), \cap, \cup, ^c, \emptyset, A)$ is a Boolean algebra. Scott [1976] uses $A = \mathbb{N}$ to discuss theory of computable functions.

For a distributive lattice, the complement of x when it exists, is unique. In the case that the complement is unique, we write $\neg x = y$ (or $x' = y$) and $\neg y = x$ (or $y' = x$). The corresponding unary operation over L, called *complementation*, introduces an analogue of logical negation into lattice theory.

Definition 4.5.47. An algebra $(H, \wedge, \vee, \rightarrow, 0, 1)$ is a ***Heyting algebra*** if

1. $(H, \wedge, \vee, 0, 1)$ is a bounded lattice (Definition 4.5.32);
2. $(H, \wedge, \vee)$ is distributive (Definition 4.5.39);
3. $x \rightarrow x = 1$
4. $x \wedge (x \rightarrow y) = x \wedge y$
5. $y \wedge (x \rightarrow y) = y$

6. $x \to (y \wedge z) = (x \to y) \wedge (x \to z)$

Heyting algebras (`https://en.wikipedia.org/wiki/Heyting_algebra`) are an example of distributive lattices where some members might be lacking complements. Every element z of a Heyting algebra has, a pseudo-complement, also denoted $\neg x$. The pseudo-complement is the greatest element y such that $x \wedge y = 0$. If the pseudo-complement of every element of a Heyting algebra is in fact a complement, then the Heyting algebra is in fact a Boolean algebra.

For some applications the distributivity condition is too strong, and the following weaker property is often useful.

Definition 4.5.48 (`https://en.wikipedia.org/wiki/Modular_lattice`)**.** A lattice $(L, \vee, \wedge)$ is ***modular*** if, for all elements $a, b, c \in L$, the following identity holds:

$$(a \wedge c) \vee (b \wedge c) = ((a \wedge c) \vee b) \wedge c. \quad \text{(Modular identity)}$$

This condition is equivalent to the following axiom:

$$a \preceq c \to a \vee (b \wedge c) = (a \vee b) \wedge c. \quad \text{(Modular law)}$$

Theorem 4.5.49. *A lattice is modular iff it does not have a sublattice isomorphic to N_5.*

The lattice theory is related to mathematics in the following way:

distributive: logic and set theory

modular: algebra, combinatorics, geometry

§4.5.2 Stone Duality

In 1934, Stone **related** the **algebraic and ordering mathematical object** — lattice — to the **geometric mathematical object**, topological space, defined below.

Definition 4.5.50. A ***topological space*** is a pair $(X, \mathscr{O}(X))$ where $\mathscr{O}(X) \subset \mathscr{P}(X)$ (Definition 4.1.1) which satisfies

- $\bigcup_{i \in I} O_i \in \mathscr{O}(X)$ (Definition 4.1.15), where I is arbitrary indexed set (Definition 4.1.14)
- $\bigcap_{i=1}^{n} O_i \in \mathscr{O}(X)$ where n is a natural number (Definition 4.1.15).

The elements O_i of $\mathscr{O}(X)$ are called *open sets* (of X). The complement of an open set, O_i^c, is called *closed sets*. Note that $\bigcup_{i \in \emptyset} O_i = \emptyset$ and $\bigcap_{i=0}^{0} O_i = X$ are both open and closed.

A subset of X which is both open and closed is called a ***clopen*** set.

Topological spaces can be classified according to `https://en.wikipedia.org/wiki/Separation_axiom` leading to the following classes of topological spaces.

Definition 4.5.51. Let X be a topological space and $x, y \in X$, $x \neq y$.

x, y is said to be ***topologically distinguishable*** if there is an open set $U \subset X$ such that $x \in U$ but $y \notin U$.

x, y is said to be ***separated*** if there are open sets $U_1, U_2 \subset X$ such that $x \in U_1$, $y \in U_2$ and either $x \notin \overline{U_2}$ or $y \notin \overline{U_1}$.

- X is T_0, or ***Kolmogorov***, if any two distinct points x, y in X are topologically distinguishable.
- X is R_0, or *symmetric*, if any two topologically distinguishable points in X are separated.

- X is T_1, or *accessible* or *Fréchet*, if any two distinct points in X are separated. Equivalently, every single-point set is a closed set. Thus, X is T_1 iff it is both T_0 and R_0.
- X is R_1, or *preregular*, if any two topologically distinguishable points in X are separated by neighbourhoods. Every R_1 space is also R_0.
- X is ***Hausdorff***, or T_2 or *separated*, if any two distinct points in X are separated by neighbourhoods. Thus, X is Hausdorff iff it is both T_0 and R_1. Every Hausdorff space is also T_1.
- X is $T_{2\frac{1}{2}}$, or ***Urysohn***, if any two distinct points in X are separated by closed neighbourhoods. Every $T_{2\frac{1}{2}}$ space is also Hausdorff.
- X is *completely Hausdorff*, or *completely* T_2, if any two distinct points in X are separated by a continuous function. Every completely Hausdorff space is also $T_{2\frac{1}{2}}$.
- X is ***regular*** if, given any point x and closed set F in X such that x does not belong to F, they are separated by neighbourhoods. Every regular space is also R_1.
- X is *regular Hausdorff*, or T_3, if it is both T_0 and regular. Every regular Hausdorff space is also $T_{2\frac{1}{2}}$.
- X is *completely regular* if, given any point x and closed set F in X such that x does not belong to F, they are separated by a continuous function. Every completely regular space is also regular.
- X is *Tychonoff*, or $T_{3\frac{1}{2}}$, *completely* T_3, or *completely regular Hausdorff*, if it is both T_0 and *completely regular*. Every Tychonoff space is both regular Hausdorff and completely Hausdorff.
- X is *normal* if any two disjoint closed subsets of X are separated by neighbourhoods. (In fact, a space is normal if and only if any two disjoint closed sets can be separated by a continuous function; this is Urysohn's lemma.)
- X is *normal regular* if it is both R_0 and normal. Every normal regular space is also completely regular.
- X is *normal Hausdorff*, or T_4, if it is both T_1 and normal. Every normal Hausdorff space is also both Tychonoff and normal regular.
- X is *completely normal* if any two separated sets are separated by neighbourhoods. Every completely normal space is also normal.
- X is *completely normal Hausdorff*, or T_5 or *completely* T_4, if it is both completely normal and T_1. Every completely normal Hausdorff space is also normal Hausdorff.
- X is *perfectly normal* if any two disjoint closed sets are precisely separated by a continuous function. Every perfectly normal space is also both completely normal and completely regular.
- X is *perfectly normal Hausdorff*, or T_6 or *perfectly* T_4, if it is both perfectly normal and T_0. Every perfectly normal Hausdorff space is also completely normal Hausdorff.

<table>
<tr><th></th><th>Separated</th><th>Separated by neighbourhoods</th><th>Separated by closed neighbourhoods</th><th>Separated by function</th><th>Precisely separated by function</th></tr>
<tr><td>Distinguishable points</td><td>Symmetric</td><td>Preregular</td><td></td><td></td><td></td></tr>
<tr><td>Distinct points</td><td>T_1</td><td>Hausdorff, T_2</td><td>Urysohn, $T_{2\frac{1}{2}}$</td><td>Completely Hausdorff</td><td>Perfectly Hausdorff</td></tr>
<tr><td>Closed set and point outside</td><td>Symmetric</td><td colspan="2">Regular</td><td>Completely regular</td><td>Perfectly normal</td></tr>
<tr><td>Disjoint closed sets</td><td>always</td><td colspan="3">Normal</td><td>Perfectly normal</td></tr>
<tr><td>Separated sets</td><td>always</td><td colspan="3">Completely normal</td><td>discrete space</td></tr>
</table>

Definition 4.5.52 (https://en.wikipedia.org/wiki/Separation_axiom)**.** A topological space X is ***sober*** if for every closed set C that is not the (possibly nondisjoint) union of two smaller closed sets, there is a unique point p such that the closure of $\{p\}$ equals C. More briefly, every irreducible closed set has a unique generic point.

Theorem 4.5.53. *Any Hausdorff space must be sober, and any sober space must be T_0.*

Definition 4.5.54. $(\mathscr{O}(X), \subset)$ is called the ***locale*** of the topological space $\mathscr{O}(X)$.

A locale is a distributive lattice and can be extended to a complete lattice called a frame.

Definition 4.5.55. A ***frame*** is a complete lattice $(L, \wedge, \vee, 0, 1)$ such that

$$u \wedge (\bigvee S) = \bigvee_{v \in S} (u \vee v)$$

for any $u \in L$ and $S \subset L$.

Definition 4.5.56. A topological space X is a *Stone space* or a *profinite set* if X is compact and totally separated, i.e. for any $x, y \in X$, if $x \neq y$ then there is a clopen $K \subset X$ such that $x \in K$ and $y \notin K$.

The Stone duality relates algebraic objects to geometric objects. This can be regarded as the relation between "equations" and "solutions". The rest of this section is taken from https://mathoverflow.net/questions/390085/analytical-origins-of-the-stone-duality.

Given an algebra-like object A, we assign to it its **poset of ideals** (typically defined as kernels of homomorphisms $A \to B$), which is interpreted as the locale (Definition 4.5.54) of open sets of a space-like object S. From any locale one can canonically extract a topological space, and this is the topological space S produced in many classical Stone-type dualities. The points of S are ideals corresponding to morphisms $A \to k$, where k is often a particularly simple algebra. These often turn out to be maximal ideals in A.

Conversely, given a space-like object S, we assign to it the algebra of morphisms $S \to k$, where k is often the "same" algebra k as above, only this time its underlying object is a space, not just a set.

A very incomplete list of examples of Stone duality from general topology, measure theory, differential geometry, algebraic geometry, and complex geometry is shown in the table below.

algebra	homomorphism	k	ideal	**space**	maps
Boolean algebra	homomorphism	$\mathbb{Z}/2$	ideal	Stone space (profinite set or compact totally disconnected Hausdorff space)	continuous map
complete Boolean algebra	complete homomorphism	$\mathbb{Z}/2$	closed ideal	compact extremally disconnected Hausdorff	open continuous map
localizable Boolean algebra	complete homomorphism	$\mathbb{Z}/2$	closed ideal	hyperstonean space	open continuous map
localizable Boolean algebra	complete homomorphism	$\mathbb{Z}/2$	closed ideal	compact strictly localizable enhanced measurable space	measurable map

distributive lattice				spectral/coherent space (a topological space homeomorphic to spectrum of a commutative ring; projective limit of finite T_0 spaces)	
frame	frame homomorphism			locale (complete Heyting algebra)	localic map
commutative von Neumann algebra	normal *-homomorphism	$\mathbb{C}$	closed *-ideal	compact strictly localizable enhanced measurable space	measurable map
commutative unital C^*-algebra	*-homomorphism	$\mathbb{C}$	closed *-ideal	compact Hausdorff space	continuous map
commutative algebra over k	homomorphism	k	ideal	coherent space / affine scheme	continuous map / morphism of schemes
finitely generated germ-determined C^∞-ring	C^∞-homomorphism	$\mathbb{R}$	germ-determined ideal	smooth locus (e.g., smooth manifold)	smooth map
finitely presented complex EFC-algebra	EFC-homomorphism	$\mathbb{C}$	ideal	globally finitely presented Stein space	holomorphic map

According to computer scientists, geometrical objects are related to computational objects as follows [Abramsky, 1991].

Geometrical Property/Object	Computational Property/Object
Space	Logical Theory
Point	Model of the theory
Open sets	Propositional formula, Semidecidable
Clopen	Decidable
Sheaf	Predicate formula
Continuous map	Transformation of models that is definable within geometric logic

§4.5.3 Denotational Semantics

In computer science, there are three styles of formal semantics:

Axiomatic Semantics Meanings for program phrases defined indirectly via the *axioms and rules of some logic of program properties*. It is used to check the compatibility of types (properties) in program specification rather than "evaluation" (meanings).

Operational Semantics Meanings for program phrases defined in terms of the *steps of computation* they can take during program execution.

They tell us **how** a computer program **run**.

Denotational Semantics Meanings for program phrases defined abstractly as elements of some suitable mathematical structure. (`https://en.wikipedia.org/wiki/Denotational_semantics`)

They tell us **what** a computer program **compute**.

In denotational semantics, it tries to model the data types of a programming language as sets and programs operating on the data types as functions (Definition 4.1.13) between sets.

Example 4.5.57 (David A. Schmidt (2012): Programming Language Semantics)**.** For a language with only numbers and addition:

$$E ::= N \mid E_1 + E_2 \qquad N = \{0, 1, 2, ...\}$$

Here, E_i stand for expressions of summation.

The (structural) operational semantics are

$$N_1 + N_2 \Rightarrow N' \qquad \frac{E_1 \Rightarrow E_1'}{E_1 + E_2 \Rightarrow E_1' + E_2} \qquad \frac{E_2 \Rightarrow E_2'}{N + E_2 \Rightarrow N + E_2'}$$

This first computation rule expresses the computation of addition. The second computation says that if the left operand of an addition expression can be rewritten, then do it. The third computation rule says that if the right operand of an addition expression can be rewritten and the left operand is already a numeral (completely evaluated), then rewrite the right operand.

Working together, the three rules force left-to-right evaluation.

Applying it to the expression $(1 + 2) + (4 + 5)$, we have

$$\underbrace{\frac{1 + 2 \Rightarrow 3}{(1 + 2) + (4 + 5) \Rightarrow 3 + (4 + 5)}}_{\text{rule 2}} \qquad \underbrace{\frac{4 + 5 \Rightarrow 9}{3 + (4 + 5) \Rightarrow 3 + 9}}_{\text{rule 3}} \qquad \underbrace{3 + 9 \Rightarrow 12}_{\text{rule 1}}$$

Example 4.5.58 (David A. Schmidt (2012): Programming Language Semantics)**.** For a language L with only numbers and addition:

$$E ::= N \mid E_1 + E_2 \qquad N = \{0, 1, 2, ...\}$$

Here, E_i stand for expressions of summation.

The denotational semantics is a mathematical model for the language L:

$$\begin{aligned} \mathcal{E} &: Expresssion \to Nat \\ \mathcal{E}[\![\mathrm{N}]\!] &= \mathrm{N} \\ \mathcal{E}[\![E_1 + E_2]\!] &= plus(\mathcal{E}[\![E_1]\!], \mathcal{E}[\![E_2]\!]) \end{aligned}$$

Applying it to the expression $(1 + 2) + (4 + 5)$, we have

$$\begin{aligned} \mathcal{E}[\![(1 + 2) + (4 + 5)]\!] &= plus(\mathcal{E}[\![(1 + 2)]\!], \mathcal{E}[\![(4 + 5)]\!]) \\ &= plus(plus(\mathcal{E}[\![1]\!], \mathcal{E}[\![2]\!]), plus(\mathcal{E}[\![4]\!], \mathcal{E}[\![5]\!])) \\ &= plus(3, 9) = 12 \end{aligned}$$

Example 4.5.59. The programming languages based on mathematical theory are listed below:

- Standard ML (Topic 1) : For the semantics, refer to "The Definition of Standard ML" document, "An ASM Dynamic Semantics for Standard ML" (operational semantics).
- Haskell : For the semantics, refer to Karl-Filip Faxén (2002)'s "A static semantics for Haskell"
- Scheme : Jacob Matthews, Robert Bruce Findler (2007)'s "An operational semantics for Scheme".
- Common Lisp : Steven S. Muchnick, Uwe F. Pleban's "A Semantic Comparison of Lisp and Scheme"
- Scala : Foundations for Scala: Semantics and Proof of Virtual Types (2006)

§4.5.4 Domain Theory

To provide a denotational semantics to a (programming) language L, computer scientists found that sets and relations are not convenient to work with function recursion and type recursion. Dana Scott (1969, 1972) proposed to use *domains*, which can be regarded a generalisation of complete lattice and monotone function [Abramsky, 1991] and provides a characterisation of recursive functions using fixed point theorem (Theorem 4.5.66).

According to `https://ncatlab.org/nlab/show/domain%20theory`, in computer science, domain theory [Cartwright et al., 2016] is concerned with the problem of finding a viable denotational semantics for certain theories of computability (such as the untyped lambda calculus): this resists straightforward interpretations in terms of plain functions between sets but does have interpretation in terms of monotone functions between partially ordered sets (certain lattices).

In computer science, a (programming) language is related to domain theory in the following way.

Language	Denotational Semantics
Type σ	Domain $D_\sigma = [\![\sigma]\!]$
Term $x_1 : \sigma_1, ..., x_n : \sigma_n \vdash M : \tau$	Continuous function $[\![M]\!] : [\![\sigma_1]\!] \times ... \times [\![\sigma_n]\!] \to [\![\tau]\!]$

According to Abramsky [1991], the stone duality provides a framework to interpret $\Gamma \models \phi$.

$\Gamma \models \phi$	Γ	ϕ
Topological view	Points	Open sets
Logical view	Models	Formulas (quantified statements, etc.)
Computer Science view	(Denotations of) computational processes	(extensions of) specifications
Denotational view, $[\![\Gamma]\!] \in [\![\phi]\!]$	$[\![\Gamma]\!]$ is a "point" in a mathematical space D_σ	$[\![\phi]\!] \subset D_\sigma$ where σ is the type of Γ

The activities of the program development process are associated with $\Gamma \models \phi$ as follows Abramsky [1991]:

- **Program specification**: the task of defining properties of ϕ to be satisfied by the program.
- **Program synthesis**: the task of finding Γ so that given ϕ is/are "true".
- **Program verification**: the task of proving that $\Gamma \models \phi$.

Definition 4.5.60. A subset X of a poset $(A, \preceq)$ is ***directed*** if every finite subset of X has an upper bound in X.

A directed subset of X is ***progressive*** if it does not contain a maximum element.

A directed subset of X is a ***chain*** if it is totally ordered (Definition 4.5.12), i.e. $x \preceq y$ or $y \preceq x$ for $x, y \in X$.

Definition 4.5.61. A ***(directed) complete partial order***, abbreviated **cpo**, is a poset $(A, \preceq)$, such that every directed subset has a least upper bound in A.

A **cpo** $(D, \preceq_D)$ is called a ***pointed cpo*** if it has a least element $\perp \in D$ such $\forall y \in D$, $\perp \preceq_D y$.

Definition 4.5.62. Let D_1, D_2 be **cpo**s. A function $f : D_1 \to D_2$ is ***(Scott) continuous*** if for all chains $C \subset D_1$,

$$f(\bigvee_{D_1} C) = \bigvee_{D_2} \{f(c) \ : \ c \in C\}.$$

Definition 4.5.63. The ***Scott topology*** on a **cpo** D, $\sigma(D)$, is given by all subsets $U \subset D$ satisfying

- $U = \{x \in D \ : \ \exists y \in U, y \preceq x\}$ (U is closed downwards under $\preceq$);
- if S a directed set and $S \subset U$, then $\vee S \in U$ (U is closed under suprema of directed sets).

Theorem 4.5.64. *Any continuous function is monotonic (Definition 4.5.17) (and preserves joins of ascending sequences).*

Continuous functions allow us to understand how infinite objects can be approximated by finite programs (without "abrupt changes").

The fixed point semantics give a mathematical explanation for loops and recursions in programming languages.

Definition 4.5.65. Let $(D, \preceq)$ be a poset and f be a function from D to D.

- $d \in D$ is a ***fixed point*** (or *fixpoint* in computer science) of f if $f(d) = d$.
- d is the ***least fixed point*** if for all $d' \in D$, $f(d') = d' \Rightarrow d \preceq d'$.

Theorem 4.5.66 (Knaster-Tarski (fixpoint) Theorem)**.** *Let D be a pointed **cpo**. Any continuous function $F : D \to D$ has a least fixed point given by*

$$fix(F) = \vee\{F^i(\perp) \ : \ i \geq 0\}.$$

Let $[D \to D]$ be all continuous function on D. The assignment

$$fix : [D \to D] \to D, \quad f \mapsto \vee_{n \in \mathbb{N}} f^n(\perp)$$

is continuous.

Definition 4.5.67. A ***finitary basis $\boldsymbol{B}$*** is a poset $(A, \preceq)$, such that A is countable and every finite consistent subset $X \subset A$ has a least upper bound in $\boldsymbol{B}$.

We call the elements of a finitary basis $\boldsymbol{B}$ propositions because they can be interpreted as logical assertions about domain elements. In propositional logic, the least upper bound of a set of propositions is the conjunction of all the propositions in the set.

Definition 4.5.68. For finitary basis $\boldsymbol{B} = (A, \preceq)$, a subset I of A is an ***ideal over $\boldsymbol{B}$*** if

1. I is downward closed: $e \in I \to (\forall b \in A, b \preceq e \to b \in I)$
2. I is closed under least upper bounds on finite subsets (conjunction).

Definition 4.5.69. Let $\boldsymbol{B}$ be a finitary basis. The ***(constructed) domain $D_{\boldsymbol{B}}$ determined by $\boldsymbol{B}$*** is a poset $(D, \preceq_D)$ where D is the set of all ideals I over $\boldsymbol{B}$ and $\preceq_D$ is the subset relation. We will frequently write D instead of $D_{\boldsymbol{B}}$.

Definition 4.5.70. An element e of a cpo $\mathcal{D} = (D, \preceq)$, is ***finite*** if for every directed subset X of D, $e = \bigsqcup X$ implies $e \in X$. The set of finite elements in a cpo $\mathcal{D}$ is denoted $\mathcal{D}^0$.

Definition 4.5.71. Two posets $\boldsymbol{A} = (A, \preceq_A)$ and $\boldsymbol{B} = (A, \preceq_A)$ are ***isomorphic***, denoted $\boldsymbol{A} \approx \boldsymbol{B}$, if there exists a bijection (Definition 4.1.13) $i : A \to B$ that preserves the partial order:

$$a \preceq_A b \Leftrightarrow i(a) \preceq_B i(b).$$

for all $a, b \in A$.

A cpo D is called a *continuous domain* if it has a basis and is called an *algebraic domain* if it has a basis of compact elements. We say D is ω-*continuous* if there exists a countable basis and we call it ω-*algebraic* if $K(D) := \{x \in D \ : \ x \text{ compact}\}$ is a countable basis.

Definition 4.5.72. A cpo $\mathcal{D} = (D, \preceq)$ is a ***domain*** if

- $\mathcal{D}^0$ forms a finitary basis under the partial order $\preceq$ restricted to $\mathcal{D}^0$, and
- $\mathcal{D}$ is isomorphic to the domain $\mathcal{E}$ determined by $\mathcal{D}^0$.

In other words, a domain is a poset that is isomorphic to a constructed domain.

Example 4.5.73.

- All finite posets are domains.
- Every complete lattice is a domain.

Theorem 4.5.74. *Continuous domains eqipped with the Scott-topology are sober spaces.*

If we are working with functions that takes a number and returns a number, a directed graph representing the iteration

$$x \mapsto f(x) \mapsto f^2(x) \mapsto \cdots$$

may be enough. However, the "domain theory" allow us to work with higher-order programming languages, i.e. programming languages that have functions of functions of functions ...

Summary of Logical Frameworks

Logical System	Syntax	Semantics	Link
Propositional Logic (Topic 1a, Topic 1b)	• Atomic Statements & Compound Statements (Formulas) • Rules of Inferences ($\to$I, $\to$E, $\wedge$I, $\wedge$E,$\vee$I, $\vee$E, $\neg$I, $\neg$E, $\bot$E) • $\phi_1, ...\phi_n \vdash \psi$ where ϕ_i, ψ are formulas	• Truth assignment v • ϕ is satisfiable: $v(\phi) = \text{T}$ or $v \models \phi$ • All assignments (combinatorial explosion) • Logical Equivalence & Laws • DNF, CNF • Logical Implications & Laws	Theorems 1.7.10 (soundness) $\vdash \psi$ implies $\models \psi$ Theorems 1.7.11 (completeness) $\models \psi$ implies $\vdash \psi$
First Order Predicate Logic (Topic 2a, Topic 2b)	• Terms (include "type" variables) & Formulas • Rules of Inferences ($\to$I, $\to$E, $\wedge$I, $\wedge$E,$\vee$I, $\vee$E, $\neg$I, $\neg$E, $\bot$E, $\forall$I, $\forall$E, $\exists$I, $\exists$E) • $\phi_1, ...\phi_n \vdash \psi$ where ϕ_i, ψ are sentences	• Model M, equality $=$ • ϕ is satisfiable: $M \models \phi$ • All models • Logical Equivalence & Laws • Prenex Normal Form (covered), CNF (not covered) • Logical Implications & Laws	Theorems 1.12.3 $\vdash \phi$ iff $\models \phi$ (Gödel's Completeness Theorem) Lean 4 Prover
Satisfiability Modulo Theory (Section 1.12)	• Variables are associated to "sorts" • Signature Σ: A set of predicate (formula) and function (term) symbols, each with associated arity. • Sentences = Formulas without free variables. • Σ-theories = A set of sentences over a signature Σ	• Structure M of signature Σ • ϕ is satisfiable: $M \models \phi$ • Some theories can be combined. Refer to page 67 of Topics 1+2 notes.	SMT Solvers (e.g. Z3 prover)
A consistent formal system F within which a certain amount of elementary arithmetic can be carried out.	Peano Arithmetic (PA) Axioms: • $\forall x(S(x) \neq 0)$ • $\forall x \forall y(S(x) = S(y) \to x = y)$ • $\forall x(x + 0 = x)$ • $\forall x \forall y(x + S(x) = S(x + y))$ • $\forall x(x \cdot 0 = 0)$ • $\forall x \forall y(x \cdot S(y) = (x \cdot y) + x)$ • $\forall y_1 \cdots \forall y_k(P(0) \wedge \forall x(P(x) \to P(S(x))) \to \forall x P(x))$, P is a predicate with free variables $x, y_1, ...y_k$. Proving Tactics (?): • Mathematical Induction • Direct Proof • Contrapositive Proof • Proof by Contradiction	Model Theory???	First Gödel's Incompleteness Theorem: There is some ϕ such that $\Gamma_{\text{PA}} \models \phi$ but $\Gamma_{\text{PA}} \nvdash \phi$ [Raatikainen, 2025]
Programming Language	• Specification • Lexical Analysis (?) • Grammar in Backus Naur Form (?)	• Program • Operational Semantics • Denotational Semantics	Program $\vDash$ Specification

Exercise with Past Year Questions

Only 2021 questions are set by me. The rest are by other lecturers.

UECM1304 May 2021 Semester

Example 4.5.75 (Final May 2021 Sem, Q4 (during MCO)). (a) Let $S = \{a, b, c, d, e\}$. A relation R on S is given by $R = \{(a, c), (b, a), (b, d), (d, b), (e, a)\}$.

(i) Write down the matrix representation M_R of the relation R. (2 marks)

Solution:

	a	b	c	d	e
a	0	0	1	0	0
b	1	0	0	1	0
c	0	0	0	0	0
d	0	1	0	0	0
e	1	0	0	0	0

..... [2 marks]

(ii) Explain whether the relation R is symmetric. (1 mark)

Solution: Since the matrix representation of R is not symmetric, the relation R is not symmetric. ..... [1 mark]

(iii) Use Warshall's algorithm to find the matrix representation of the transitive closure $cl_{trn}(R)$ of R in part (i) and then write down $cl_{trn}(R) \setminus R$. (3 marks)

Solution:

	a	b	c	d	e
a	0	0	1	0	0
b	1	1	1	1	0
c	0	0	0	0	0
d	1	1	1	1	0
e	1	0	1	0	0

..... [2.7 marks]

$cl_{trn}(R) \setminus R = \{(b, b), (b, c), (d, a), (d, c), (d, d), (e, c)\}$. ..... [0.3 mark]

(b) Show that the relation R on $\mathbb{Z}$ defined by $xRy := 2x + 5y \equiv 0 \pmod 7$ is an equivalence relation and write down the equivalence classes $\mathbb{Z}/R$. (3 marks)

Solution: First, we show that it is reflexive: For every $x \in \mathbb{Z}$,

$$2x + 5x = 7x \equiv 0 \pmod 7 \Rightarrow xRx. \qquad \text{[0.5 mark]}$$

Next, we show that it is symmetric: For any x, $y \in \mathbb{Z}$, if xRy, i.e. $2x + 5y \equiv 0 \pmod 7$, then

$$(2-7)x + (5-7)y \equiv 0 \Rightarrow -(-5x - 2y) \equiv -0 \Rightarrow 2y + 5x \equiv 0 \pmod 7 \Rightarrow yRx. \qquad \text{[1 mark]}$$

Finally, we show that R is transitive: For any $x, y, z \in \mathbb{Z}$, if xRy and yRz, then

$$\begin{aligned} &2x + 5y \equiv 0 \wedge 2y + 5z \equiv 0 \\ \Rightarrow &2x \equiv -5y \equiv 2y \Rightarrow 2y + 5z \equiv 2x + 5z \equiv 0 \Rightarrow xRz. \end{aligned} \qquad \text{[1 mark]}$$

The equivalence classes are $7\mathbb{Z}$, $7\mathbb{Z} + 1$, ..., $7\mathbb{Z} + 6$. ..... [0.5 mark]

(c) State an example of poset with finite elements and an example of poset with infinite elements which are taught in the lecture. (1 mark)

Solution: An example of poset with finite elements is $(P(S), \subset)$ where S is a finite set and $P(S)$ is the power set of S. ..... [0.5 mark]

An example of poset with infinite elements is $(\mathbb{Z}, \leq)$. ..... [0.5 mark]

[Total: 10 marks]

UECM1304 May 2024 Semester

Example 4.5.76 (Final May 2024 Sem, Q4)**.** (a) Let $A = \{2, 4, 5, 7, 10, 12, 15\}$. Define the following relation R on A:

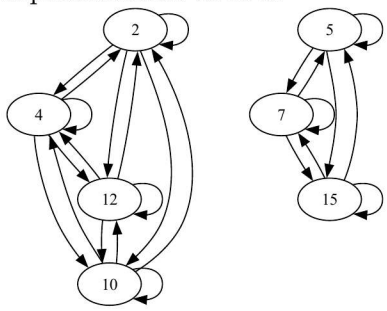
$$(a, b) \in R \text{ iff } a \equiv b \pmod 2.$$

Find R, M_R and draw the digraph if R. Explain whether R is reflexive, irreflexive, symmetric, asymmetric, antisymmetric or transitive. Give reason(s) to support your answer. (14 marks)

Solution: $R = \{(2,2), (4,4), (5,5), (7,7), (10,10), (12,12), (15,15), (2,4), (2,10), (2,12), (4,2), (4,10), (4,12), (5,7), (5,15), (7,5), (7,15), (10,2), (10,4), (10,12), (12,2), (12,4), (12,10), (15,5), (15,7)\}$

$$M_R = \begin{array}{c} \\ 2 \\ 4 \\ 5 \\ 7 \\ 10 \\ 12 \\ 15 \end{array} \begin{array}{c} \begin{array}{ccccccc} 2 & 4 & 5 & 7 & 10 & 12 & 15 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Digraph representation of R is



Since diagonal of M_R are all ones, R is reflexive.
Since $M_R = M_R^T$, R is symmetric.
It is transitive because for each sub-digraph the digraph, we see that all nodes have edges to itself and the rest.

(b) Suppose $B = \mathbb{Z}^+$. Define the following relation S on B:

$$aSb \text{ iff } b|a.$$

Prove that (B, S) is a partially order set. (8 marks).

Proof: Partially order set needs to satisfy 3 conditions (according to Definition 4.5.1):

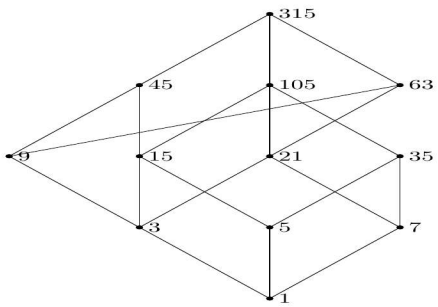
(a) reflexive: Since $a|a$, aSa.

(b) anti-symmetric: If aSb and bSa, then $b|a$ and $a|b$, i.e.for some $k_1 > 0$ and $k_2 > 0$, $a = bk_1$ and $b = ak_2$. This implies $a = (ak_2)k_1$. Since $B = \mathbb{Z}^+$, $k_1k_2 = 1$ and so $k_1 = k_2 = 1$, i.e. $a = b$.

(c) transitive: If aSb and bSc, then $b|a$ and $c|b$, i.e. for some $k_1 > 0$, $k_2 > 0$, $a = bk_1$ and $b = ck_2$. This means $a = ck_2k_1$, i.e. $c|a$ or cSa by definition.

(c) Draw a Hasse diagram for $(D_{315}, |)$. (3 marks)

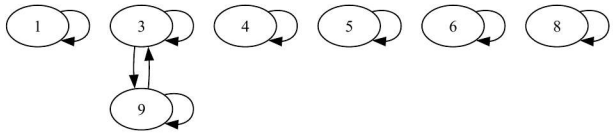
Solution: $D_{315} = \{1, 3, 9, 15, 21, 35, 45, 63, 105, 315\}$ is the set of all divisors of 315 (refer to Example 4.5.22)



Example 4.5.77 (Final May 2024 Sem, Q5)**.** (a) Let $A = \{1, 3, 4, 5, 6, 8, 9\}$ and let R and S be binary relations on A such that xRy iff $2|(y-x)$ and xSy iff $3|(x-y)$. Draw the digraph for $R \cap S$. (12 marks)

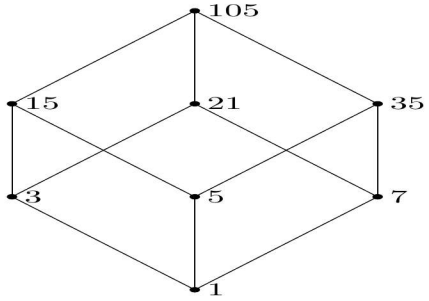
Solution: $R \cap S = \{(x, y) \in A \times A \ : \ 2|(y - x) \wedge 3|(x - y) \equiv \exists m \exists n(x - y = 2m = 3n) \equiv \exists k(x - y = 6k)\}$

The digraph of $R \cap S$ is



(b) Draw a Hasse diagram for $(D_{105}, |)$. (3 marks)

Solution: $D_{105} = \{1, 5, 3, 7, 15, 21, 35, 105\}$ is the set of all divisors of 105.



(c) Prove whether the relation "congruent modulo" is an equivalence relation on the set of all integers. Give reason to support your answer. (10 marks)

Note: Congruent modulo is an equivalence relation. The proof is given in Topic 3. Refer to the Corollary after the Theorem of Basic Arithmetic of Congruences in Topic 3.

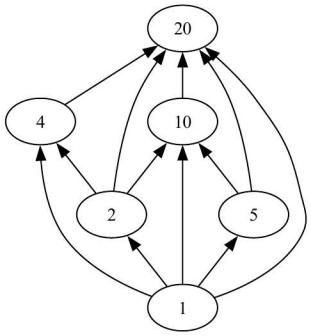
UCCM1363 Jan 2024 Semester

Example 4.5.78 (Final Jan 2024 Sem, Q3)**.** (a) List out the relation that represents the relation S defined on $\{1, 2, 4, 5, 10, 20\}$ by $xSy \Leftrightarrow \{x < y \text{ and } x \text{ divides } y\}$. (5 marks)

Solution: $S = \{(1,2), (1,4), (1,5), (1,10), (1,20), (2,4), (2,10), (2,20), (4,20), (5,10), (5,20), (10,20)\}$

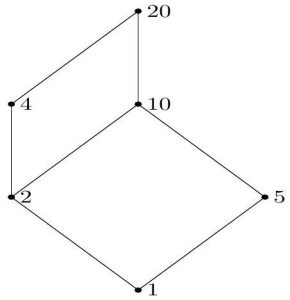
(b) Plot the directed acyclic graph that represents the relation S in (a). (6 marks)

Solution:



(c) Plot the Hasse diagram that represents the relation S in (b). (5 marks)

Solution: Note that according to Wikipedia, Hasse diagram can also be used for strict partial ordering, i.e. both non-reflexive and anti-symmetric are not satisfied.



(d) Based on question (a) above, answer the following as True or False and give example.

(i) If $(x, y) \in S$, then $(y, x) \notin S$, for all $x, y \in S$ (3 marks)

(ii) $(x, x) \in S$, $x \in S$ (3 marks)

(iii) If $(x, y) \in S$, and $(y, z) \in S$ then $(x, z) \in S$, for all $x, y, z \in S$ (3 marks)

Solution:

(i) ..... **True**

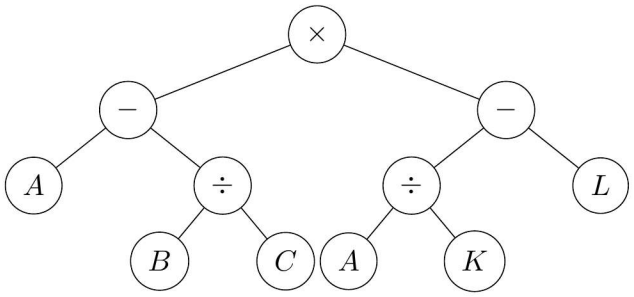
(ii) ..... **False**

(iii) ..... **True**

Example 4.5.79 (Final Jan 2024 Sem, Q4)**.** (a) Construct a label tree representing the following algebraic expression.

$$(A - B \div C) \times (A \div K - L).$$ (8 marks)

Solution: Using what you learn in Topic 1, following the precedence of operators, the label tree can be easily obtained:



(b) Develop the prefix and postfix for (a). (6 marks)

Solution:

Prefix: × − A ÷ B C − ÷ A K L

Postfix: A B C ÷ − A K ÷ L − ×

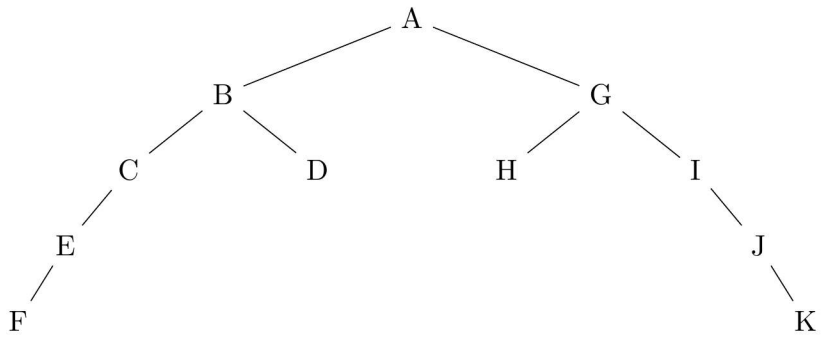
(c) Application of algorithm for the *inorder* and *preoder* traversals of a binary tree with 11 nodes yields the following sequences of nodes.

INORDER : F E C B D A H G I J K

PREORDER : A B C E F D G H I J K

Construct the diagram of the binary tree. (11 marks)

Solution: This is a detective question where we need to use PREORDER to determine each "root" for each binary sub-tree.



References

S. Abramsky. Domain theory in logical form*. *Annals of Pure and Applied Logic*, 51(1):1–77, 1991. ISSN 0168-0072. doi: https://doi.org/10.1016/0168-0072(91)90065-T. URL `https://www.sciencedirect.com/science/article/pii/016800729190065T`.

R. Cartwright, R. Parsons, and M. AbdelGawad. Domain theory: An introduction, 2016. URL `https://arxiv.org/abs/1605.05858`.

V. K. Garg. *Introduction to Lattice Theory with Computer Science Applications*. John Wiley & Sons, Ltd, 2015.

G. Heiser. *The seL4 Microkernel: An Introduction*. The seL4 Foundation, revision 1.4 of 2025-01-08 edition, 2025.

P. Raatikainen. Gödel's Incompleteness Theorems. In E. N. Zalta and U. Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2025 edition, 2025.

D. Scott. Continuous lattices. In F. W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, pages 97–136, Berlin, Heidelberg, 1972. Springer Berlin Heidelberg. ISBN 978-3-540-37609-5.

D. Scott. Data types as lattices. *SIAM Journal on Computing*, 5(3):522–587, 1976. doi: 10.1137/0205037. URL `https://doi.org/10.1137/0205037`.