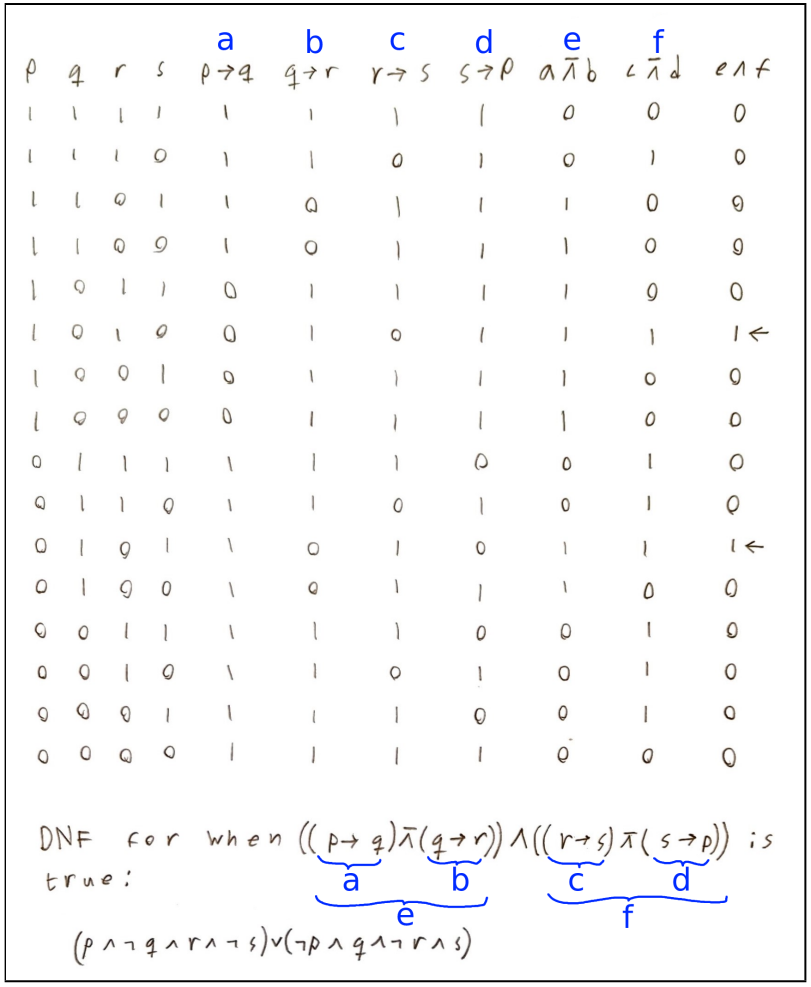
DNF solving

You can use a truth table to solve DNF problems. Start by adding a column for each literal with all combinations of 1s and 0s. Then you add columns to your truth table for each subexpression until you've expressed the entire expression. The rows containing a 1 in the final column are your answer. See p.12 in Definitions for *basic logic connectives*.



Intervals

In this problem, functions are applied to intervals, with intervals as output.

For the following sets of numbers, specify the smallest and the largest numbers, write NONE if there is no smallest or largest number, or EMPTY (in one of the two columns) if the set is empty.

All intervals are supposed to be intervals in the real numbers, $\mathbb{R}$. Similarly, all relations and operators are on the real numbers, unless explicitly stated otherwise.

set	smallest element	largest element
$\{x \in \mathbb{Z} : x > 4 \wedge x < 2\}$	EMPTY	
$\{1, 2, 3, 4\}$	1	4
$]1, 4]$	NONE	4
$\bigcap_{i \in \mathbb{N}^+} \left[0, \frac{1}{i}\right] = \{0\}$	0	0
$\bigcap_{i \in \mathbb{N}^+} \left]0, \frac{1}{i}\right] = \emptyset$	EMPTY	
$\bigcap_{i \in \mathbb{N}^+} \left[\frac{-1}{i}, \frac{1}{i}\right] = \{0\}$	0	0
$\bigcap_{i \in \mathbb{N}^+} \left]\frac{-1}{i}, 1\right] = [0, 1]$	0	1
$\text{sqrt}([0, 0.1]) = [0, \sqrt{0.1}]$	0	$\sqrt{0.1}$
$\text{sqrt}[[0, 0.1]] = [0, 1[$	0	NONE
$\text{sqrt}[]0, 0.1]] =]0, 1[$	NONE	NONE
$\bigcup_{i \in \mathbb{N}^+} \left]\frac{1}{i+1}, \frac{1}{i}\right[=]0, 1[$	NONE	NONE
$\text{inv}(]0, 0.1]) = [10, +\infty[$	10	NONE
$\text{inv}[]0, 0.1]] =]0, 0.1] \cup [10, +\infty[$	NONE	NONE

$\text{sqrt} : \mathbb{R}^+ \longrightarrow \mathbb{R}^+$ is the positive square root function, i.e. for every non-negative real number a, $\text{sqrt}(a)$ is the non-negative real number such that $a = \text{sqrt}(a) \cdot \text{sqrt}(a)$.

$\text{inv} : \mathbb{R} \setminus \{0\} \longrightarrow \mathbb{R} \setminus \{0\}$ is the inversion function, defined by $\text{inv} : r \mapsto \frac{1}{r}$.

Below is a explanation of how to think about intervals in more detail.

When looking at intersections of an infinite number of sets, it is important to keep in mind that any value that is an element of the intersection must be an element of each and every one of those sets.

Take for example $\bigcap_{i \in \mathbb{N}^+} \left[0, \frac{1}{i}\right]$. If that intersection contained any positive real number in addition to 0, it would mean that there is a real number $r > 0$, such that $r \in \left[0, \frac{1}{i}\right]$ for every $i \in \mathbb{N}^+$. It's easy to see that this cannot be the case: for any $r > 0$, there is some natural number $k > \frac{1}{r}$, which means that $\frac{1}{k} < r$, and so $r \notin \left[0, \frac{1}{k}\right]$, and so r cannot be in the intersection.

Injective and surjective

This is a simpler example of injective and surjective functions. The reason why this question is simpler is because we can define the domain and codomain separately.

Define two sets A and B, as well as a function $f : A \longrightarrow B$, such that f is **surjective** and **not injective**.

$A = \{a, b\}$ $B = \{x\}$ $f : a \mapsto x$

Of course, if you already found the answer to the next question, you could "reuse" it here by simply choosing to make A and B the same set.

The idea was to start with an easier question to get you to think about surjectivity and injectivity in a simpler setting first, before tackling the harder problem.

When something should be surjective but **not** injective and the *codomain* = *domain* we need to utilize infinity. Below, any element in the codomain could be reached from the domain by adding one to it. Meaning it is surjective but we also point to the same element twice, 0 in this case.

Define **one** set A, as well as a function $f : A \longrightarrow A$, such that f is **surjective** and **not injective**.

$A = \mathbb{N}$ $\quad f : a \mapsto \begin{cases} 0 & \text{if } a = 0 \\ a - 1 & \text{otherwise} \end{cases}$

The crux here is that A has to be infinite.

Suppose there is a function $f : A \longrightarrow A$ which is surjective and **not** injective, like the one you were asked to define in the previous task. This question is about a property of A (the domain and codomain of f) that implies that such a function exists, and which is also implied by the existence of such a function. (You do not need to prove this here.)

A surjective and **not** injective function $f : A \longrightarrow A$ exists if and only if

A is infinite.

(this must be a statement about the set A, and cannot involve f)

If you want to get a deeper understanding of this point, try to prove it. You can do this in two steps:
(1) You show that if A is infinite, a function exists on it that is surjective but not injective. This you can show by taking our definition of an infinite set (one that is equivalent to a proper subset of itself), and use that definition to construct such a function.

(2) Now you need to show that if such a function exists, then A is infinite. Being infinite means that A must be equinumerous to a proper subset of itself. So given a function that is surjective but not injective, you need to find a proper subset of A that is the same size as A.

If you find this confusing, have a look at the solution above, and try to figure out what a suitable proper subset of A would be, and how it is related to f.

Properties of paths, graphs and trees

Properties of a **directed graph**, worth remembering is that no property is guaranteed. We don't put any constraints on our graph meaning it could be any kind of relation, for example every edge could be symmetric.

Recall that a *directed graph* (V, E) is defined as a finite set V of vertices and a relation $E \subseteq V \times V$ between them.
This question is about the properties of that relation. In the table below, make one mark in each row for the property in the left column, depending on whether all, some, or no relations defining a graph have that property. Put the mark in the corresponding ALL box, if **all relations** defining a graph have the corresponding property, the NONE box, if **no relation** has it, and the SOME box if at least one relation does, and at least one does not.

	ALL	SOME	NONE
reflexive over V		X	
transitive		X	
symmetric		X	
antisymmetric		X	
asymmetric		X	

Graphs in our definition make no special assumptions about the relations that define them, so any (finite) relation could be a graph.

Below are the properties of a **rooted tree**. The reason it's antisymmetric and asymmetric is because if there exist (a,b) and (b,a) then there are multiple ways to reach b (ie. $a \to b$, $a \to b \to a \to b$). The reason why it's antisymmetric is the same reason it is not reflexive.

Recall that a *rooted tree* is a graph (T, R) such that, if the set T of nodes is not empty, then there is a node $a \in T$ (the root) such that for every $x \in T$ with $x \neq a$ there is exactly one path from a to x. Like V in the previous question, $R \in T \times T$ is a relation on the set of nodes. To make things simpler, for this question we only consider non-empty trees, that is $T \neq \emptyset$.

This question is about the properties of the relations defining trees. In the table below, make one mark in each row for the corresponding property in the left column, depending on whether all, some, or no relations defining a tree have that property. Put the mark in the corresponding ALL box, if **all relations** defining a tree have the corresponding property, the NONE box, if **no relation** has it, and the SOME box if at least one relation does, and at least one does not.

	ALL	SOME	NONE
reflexive over T			X
transitive		X	
symmetric		X	
antisymmetric	X		
asymmetric	X		

The situation is different for trees, which are much more specialized and constrained structures than graphs. Since we only consider non-empty trees, none of them can be reflexive. (If we allowed the empty tree, then its link-relation R would also be empty, which is reflexive over the empty set.)

But there are trees whose link relation is transitive, viz. all those of link height 0 or 1. (Make sure you understand why that is.) And there is a tree whose link relation is symmetric, namely the tree consisting of only a root, whose link relation is therefore empty, which is symmetric.

Other important definitions

Node height/depth

The **link-height** (*alias:* level) in a tree is defined recursively: that of the root is 0, and that of each of the children of a node is one greater than that of the node.

The **node-height** (*alias:* height) is defined by the same recursion, except that the node-height of the root is set to 1. Thus, for every node x, $node - height(x) = link - height(x) + 1$. As trees are usually drawn upside-down, the term 'depth' is often used instead of 'height'.

CNF

Conjunctive normal form is like disjunctive normal form but 'upside-down': the roles of disjunction and conjunction are reversed. A basic disjunction is defined to be any disjunction of (one or more) literals in which no letter occurs more than once. A formula is said to be in conjunctive normal form (CNF) iff it is a conjunction of (one or more) basic disjunctions

Minimal vs Minimum

minimal element != minimum element:

1. minimal means that there is no smaller element
2. minimum means all other elements are greater

p. 8 in slides under *well founded sets*

Edge cases for always/sometimes/never problems

If you have a graph (V, E), then you should check the edge cases where:

1. $V = \emptyset$ or $E = \emptyset$.
2. V only contains 1 node or 2 nodes.
3. E contains a connection in one direction but not back. And when it contains a connection in both directions.
4. All nodes are connected to all nodes.

Quantifiers

$\forall x \in \emptyset(\ldots) = true.$
$\exists x \in \emptyset(\ldots) = false.$

Logic operators

1. $\alpha \barwedge \beta = \neg(\alpha \wedge \beta)$ and $\neg\alpha = (\alpha \barwedge \alpha)$

Image of n-ary

When computing the image, we treat the last element of a tuple as the 'output'.

$$R(a_1, \ldots, a_{n-1}) = \{a \in A : (a_1, \ldots, a_{n-1}, a) \in R\} \tag{1}$$

Order

First, I think it's often helpful to have examples in mind. Here are 4 examples of relations for ordering.
a) The relation $\leq$ defined for $\mathbb{Z} \times \mathbb{Z}$ is a (non-strict) total order
b) The relation $<$ defined for $\mathbb{Z} \times \mathbb{Z}$ is a strict total order
c) The relation $\subseteq$ defined for $\mathcal{P}(\mathbb{N}) \times \mathcal{P}(\mathbb{N})$ is a (non-strict) partial order
d) The relation $\subset$ defined for $\mathcal{P}(\mathbb{N}) \times \mathcal{P}(\mathbb{N})$ is a strict partial order

Total order refers to all pairs of elements being comparable with regards to the order while for a partial order this is not *necessarily* the case. An order is strict if no element is comparable to itself.

In our examples, $\subseteq$ and $\subset$ are partial order (and are not total), since two elements or not necessarily comparable, e.g. $\{1, 2\}$ and $\{3\}$ are not comparable under $\subseteq$, since $\{1, 2\} \nsubseteq \{3\}$ and $\{3\} \nsubseteq \{1, 2\}$.

In a (non-strict) total order, every pair of elements are comparable, e.g. for any two numbers $a, b \in \mathbb{Z}$ it holds that $a \leq b$ or $b \leq a$ (or both if $a = b$).

In a strict total order, every pair of *distinct* elements are comparable. In our example, for any two numbers $a, b \in \mathbb{Z}$ if $a \neq b$ then $a < b$ or $b < a$.

Note that according to the definitions total orders are a special case of partial orders.

A total order is also called a linear order. The intuition behind this is that you can place elements in a line in increasing order (according to the total order).

One more observation is that when we say "poset" (or partially ordered set) we mean a pair (R, A) where R is a partial order defined for $A \times A$. So it is not that the set alone is partially ordered, but that R defines a partial order on A. Sometimes you might read that something like "the natural numbers form a totally ordered set", but this actually means that there is a relation, e.g. $\leq$, that defines a total order on $\mathbb{N}$.

Extras

Properties of composition of relations

Suppose, as previously, $A = \{n \in \mathbb{N} : 1 \leq n \leq 10\}$ and a relation
$R_3 = \{(a, b) \in A \times A : \mod (b, a) = 3\}$.

Let $T = R_3 \circ R_3^{-1}$.

Here, too, things become a lot simpler once you have explicitly constructed the extensions of the relations involved. So for reference:

$R_3 = \{(4,3), (4,7), (5,3), (5,8), (6,3), (6,9), (7,3), (7,10), (8,3), (9,3), (10,3)\}$

$R_3^{-1} = \{(3,4), (7,4), (3,5), (8,5), (3,6), (9,6), (3,7), (10,7), (3,8), (3,9), (3,10)\}$

$T = \{(3,3), (7,7), (8,8), (9,9), (10,10), (3,7), (3,8), (3,9), (3,10), (10,3), (9,3), (8,3), (7,3)\}$

1. [3 p] $\#T = 13$
2. [3 p] $T(1) = \{\}$
3. [3 p] $T(7) = \{3, 7\}$
4. [3 p] $T(A) = \{3, 7, 8, 9, 10\}$
5. [4 p] T is ... (circle those that apply)

… reflexive	TRUE	(FALSE)
… symmetric	(TRUE)	FALSE
… transitive	TRUE	(FALSE)
… antisymmetric	TRUE	(FALSE)

Recursive traversion of a tree

Suppose we have a rooted tree (T, R) with nodes T, links $R \subseteq T \times T$, and root a as well as a labeling function $\lambda : T \longrightarrow \mathbb{N}$ assigning each node in the tree a natural number.

We want to define a function $L : T \longrightarrow \mathbb{N}$ that computes for each node $n \in T$ the lowest number a node in the subtree rooted at n is labeled with (that subtree includes n itself). If the subtree consists only of n, its label $\lambda(n)$ is the lowest number.

As before, for any non-empty set S of numbers, $\min S$ is the lowest number in that set.

1. [10p] Define L using well-founded recursion. (Hint: You may use cases if you like, but it is possible to define this function without an explicit "base case.")

   $$L : n \mapsto \min\{\lambda(n)\} \cup \{L(n') : nRn'\}$$

   Note that the nRn' takes care of the "termination": if there is no child, is means there is no n', and the second set in the union above will simply be empty.

2. [8p] Define a strict partial order $\prec$ on T such that the poset $(T, \prec)$ is well-founded and your definition of L performs well-founded recursion on that poset. For all $n, n' \in T$...

   $$n' \prec n \iff nR^*n'$$

   R^* is the transitive closure of R. Specifying *only* the link relation itself would not result in a poset, since it is not a partial order.

   There are other ways of answering that question, for example using the closure of {n} under R, i.e. R[{n}]: $n' \prec n \iff n' \in R[\{n\}]$

   Many answers tried to use L to define the order. However, that does not work since, for instance, the labeling could give the same number to every node (there is nothing that would require it not to), and so there would be no way to distinguish between nodes, which a strict order would have to.

   Also, it would not make much sense to do so, since the point of this order is to demonstrate the well-definedness of L, so using L to define it would be oddly circular. However, this in itself would not make the answer wrong (only useless for its intended purpose).

   No proof is required. It is sufficient that the strict partial order is well-founded and your definition of L conforms to it.

   Hint: Make sure the partial order you define actually is one, i.e. that it has all the properties required from a strict partial order, including, for example, transitivity.

Injective functions

Suppose you have **injections** $f : A \hookrightarrow B$ and $g : A \hookrightarrow B$, as well as a **non-empty** set $S \subset A$ (note that S is a **proper** subset of A). Now let's define a function $h : A \longrightarrow B$ as follows:

$$h : x \mapsto \begin{cases} f(x) & \text{for } x \in S \\ g(x) & \text{for } x \notin S \end{cases}$$

This function is not, in general, injective.
Whether it is injective depends on the definitions of A, B, f, g, and S.

1. [5p] Give definitions for A, B, f, g, and S such that the h above is injective.

   A = {a, b} B = {x, y} f = {(a, x), (b,y)} g = {(a, x), (b,y)} S = {a}

2. [5p] Give definitions for A, B, f, g, and S such that the h above is **not** injective.

   A = {a, b} B = {x, y} f = {(a, x), (b,y)} g = {(a, y), (b,x)} S = {a}

3. [8p] Give a general formal criterion, depending only on A, B, f, g, and S (not necessarily all of them), that defines the condition under which h is injective. (Hint: Remember, f and g are already injective.)

   h is injective iff $f(S) \cap g(A \setminus S) = \emptyset$

Note: You are **not** supposed to reiterate the definition of injectivity for h, but rather give an expression involving at most A, B, f, g, and S (but **not** h) that is true if an only if they lead to an injective h.

Common mistakes on the first two subquestions included answers that defined f and g in such a way that they were not functions from A to B. Often, f was defined only on S and g on A\S. Occasionally, the image of A under f or g was not in B.

On the third subquestion, some answers missed that the criterion was supposed to be true *if and only if* (*iff*) h was injective. So, for example, if f(A) and g(A) are disjoint, then h will be injective, but that's not required: h will also be injective in cases where f(A) and g(A) aren't disjoint (for example, it could be that f=g, and in fact then h will always be injective, for any S), so that criterion is too strict.

Injections, Bijections, Infinite sets and Recursion

Suppose you have an infinite set X and in **injection** $f : X \hookrightarrow \mathbb{N}$.
Note that this implies that X and $\mathbb{N}$ are equinumerous. (Make sure you understand why that is the case.)
The goal is to use f to define a **bijection** $g : \mathbb{N} \longleftrightarrow X$.
Note that we cannot simply invert f – while injectivity guarantees that every $n \in \mathbb{N}$ **is mapped to at most once** by f, some n may not be mapped to at all. For any $n \in \mathbb{N}$, either $f^{-1}(n) = \emptyset$, i.e. n was not mapped to by f, or $f^{-1}(n) = \{x\}$, the singleton set of the one value $x \in X$ mapped to n by f. Let us call the set of all values mapped to by f by the name M, i.e. $M = f(X)$.

Example: For instance, suppose $X = \{a, b, c\}^*$, i.e. the set of all finite strings of a, b, and c. f might then be $\{(aca, 2), (bbccaa, 5), (ccc, 7), (cbabc, 12), ...\}$ (listed in order of the number mapped to, so there is no mapping to 0, 1, 3, 4, 6, 8, 9 etc.). So in this case, M would be $\{2, 5, 7, 12, ...\}$.
We define the bijection g using a helper function $h : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$.
$h(n, k)$ is the $(n + 1)^{th}$ number in M (in the usual numerical order) greater or equal to k. Since all natural numbers are greater or equal to 0, $h(n, 0)$ is simply the $(n + 1)^{th}$ number in M, which we then can use to define the bijection g as follows:

$$g : \mathbb{N} \longleftrightarrow X$$
$$n \mapsto x \text{ with } f^{-1}(h(n, 0)) = \{x\}$$

To help you understand how to define h, note that, in the example, 7 is the $(2 + 1)^{th}$, i.e. third, number greater or equal to 0 in $M = \{2, 5, 7, 12, ...\}$, but it is also the $(1 + 1)^{th}$, i.e. second, number greater or equal to, for example, 3, and the $(0 + 1)^{th}$, i.e. first, number greater or equal to 6. Therefore, in the example, the following calls to h all yield the same result:

$$h(2, 0) = h(2, 1) = h(2, 2) = h(1, 3) = h(1, 4) = h(1, 5) = h(0, 6) = h(0, 7) = 7.$$

Understanding these equivalences should give you an idea how to construct the definition of h.
So, in the case of the example, $g(2)$ will call $h(2, 0)$, resulting in 7, and then $f^{-1}(7) = \{ccc\}$, and thus $g(2) = ccc$. Similarly, $g(1) = bbccaa$, $g(0) = aca$, etc.
Define h recursively.

$$h : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$$
$$n, k \mapsto \begin{cases} k & \text{if } f^{-1}(k) = \{x\} \wedge n = 0 \\ h(n-1, k+1) & \text{if } f^{-1}(k) = \{x\} \wedge n > 0 \\ h(n, k+1) & \text{if } f^{-1}(k) = \emptyset \end{cases}$$

Many answers used $k \in M$ instead of $f^{-1}(k) = \{x\}$, and correspondingly $k \notin M$ for $f^{-1}(k) = \emptyset$, which is fine.

Recursively computing all simple cycles

Suppose you have a graph (V, E) with vertices V and edges $E \subseteq V \times V$.
As before, a *path* in this graph is a non-empty finite sequence $v_0 v_1 ... v_n \in V^*$, such that for an $i \in \{0, ..., n-1\}$ y we have $(v_i, v_{i+1}) \in E$. The number n, corresponding to the number of edges connecting the vertices in the path (and one less than the number of vertices in the sequence representing it), is called its *length*.
A *cycle* is a path of at least length 1 where the first and the last vertex are the same, so $v_0 = v_n$. A *simple cycle* is a cycle where every vertex occurs at most once, except for the first and last, which occurs exactly twice.
This task is about defining a function $C : V \longrightarrow \mathcal{P}(V^*)$ that for any vertex $v \in V$ computes **the set of all simple cycles** starting (and therefore also ending) at v.
We shall do so using a helper function $C' : V \times V^* \times V \longrightarrow \mathcal{P}(V^*)$, such that $C'(v, p, w)$ is the set of all simple cycles that (a) start (and end) at v, (b) then follow the path p, and (c) then continue with vertex w. In other words, $C'(v, p, w)$ is the set of all simple cycles that begin with vpw.
Using this, we can define C as follows (remember that ε represents the empty sequence):

$$C : V \longrightarrow \mathcal{P}(V^*)$$
$$v \mapsto \bigcup_{w \in E(v)} C'(v, \varepsilon, w)$$

Convince yourself that this results in all simple cycles starting at v if C' behaves as described above.

1. [20 p] Define C' recursively. You may find it useful to look at the *set* of all vertices occurring in a path $p \in V^*$. You can use the notation $set(p)$ for this purpose, i.e. if p is the path $v_0 v_1 ... v_n$, then $set(p)$ is the set $\{v_0, v_1, ..., v_n\}$.
   $C' : V \times V^* \times V \longrightarrow \mathcal{P}(V^*)$

$$v, p, w \mapsto \begin{cases} \{vpw\} & \text{if } v = w \\ \bigcup_{x \in E(w)} C'(v, pw, x) & \text{if } v \neq w \wedge w \notin set(p) \\ \emptyset & \text{if } v \neq w \wedge w \in set(p) \end{cases}$$

   One answer collapsed the cases into an elegant one-liner, roughly like this:

$$v, p, w \mapsto \{vpw : v = w\} \cup \bigcup_{x \in \{y \in E(w) : v \neq w \wedge w \notin set(p)\}} C'(v, pw, x)$$

2. [10 p] In order to ensure that C' terminates, we require a **well-founded strict order** $\prec$ of its arguments, such that for any (v, p, w) that C' is called on, it will only ever call itself on $(v', p', w') \prec (v, p, w)$. Define such an order $(v', p', w') \prec (v, p, w) \iff set(p') \supset set(p)$

Note that the order must rely on the *set* of symbols in the partial path p. It is true, of course, that p' is always also a prefix of p, but using the prefix property to establish the order does not work because there are infinite chains in it (in other words: sequences can get longer forever, but there are only a finite number of vertices, so if we add a new one at every step, we will eventually terminate).

Recursively computing the cost of a path

Suppose we have a function $d : E \longrightarrow \mathbb{R}^+$ from the combined connections to the positive real numbers, signifying for each $(v_1, v_2) \in E$ the time it takes to travel from v_1 to v_2 (by either road or ferry, depending on whether $(v_1, v_2) \in R$ or $(v_1, v_2) \in F$). For any $(v_1, v_2), (v_2, v_1) \in E$, it is NOT guaranteed that $d(v_1, v_2) = d(v_2, v_1)$.

Given a path $v_0 v_1 v_2 ... v_n$ of length n in E, we want to measure it regarding the total time it takes and also the number of ferry connections in the path. The function $M : \Pi \longrightarrow \mathbb{N} \times \mathbb{R}_0^+$ computes for each non-cyclic path $p \in \Pi$ (the set of all non-cyclic paths from the previous task) a pair (n, r), where n is the number of ferry connections and r the sum of the travel time of all the connections, by road or ferry, in the path.
For every $v \in V$, applying M to the empty path v results in $(0, 0)$, i.e. $\forall v \in V \ (M(v) = (0, 0))$.
Define M.
$M : \Pi \longrightarrow \mathbb{N} \times \mathbb{R}_0^+$

$$p \mapsto \begin{cases} (0,0) & \text{for } p = v \\ (n+1, r + d(v_1, v_2)) & \text{for } p = v_1 v_2 q \wedge (v_1, v_2) \in F \wedge M(v_2 q) = (n, r) \\ (n, r + d(v_1, v_2)) & \text{for } p = v_1 v_2 q \wedge (v_1, v_2) \in R \wedge M(v_2 q) = (n, r) \end{cases}$$

Hint: you might find it useful to distinguish cases where p has the form v (i.e. it is an empty path), from cases where it has the form $v_1 v_2 q$, i.e. a path with at least two vertices (i.e. of length at least 1).

This is a problem involving recursion (or otherwise iterating) over strings/finite sequences, in this case sequences representing paths. The first point here, and one that most answers got right, is the realization that one needs to distinguish between “ferry” and “road” edges connecting vertices in the path. The second is that while we need to look at the first two vertices (so we can figure out what kind of edge connects them), we must only “peel off” the first vertex, and keep recurring on the path beginning with the second. In other words, in the above formulation, the recursive call needs to be $M(v_2 q)$ rather than just $M(q)$.

Some solutions tried to get away with a non-recursive way of computing the two components of the result, which is okay if it is done correctly. Similarly, some solutions used position-wise addition on pairs, which is fine (some of you had asked me about this during the exam), e.g. like so:

$$p \mapsto \begin{cases} (0,0) & \text{for } p = v \\ (1, d(v_1, v_2)) + M(v_2 q) & \text{for } p = v_1 v_2 q \wedge (v_1, v_2) \in F \\ (0, d(v_1, v_2)) + M(v_2 q) & \text{for } p = v_1 v_2 q \wedge (v_1, v_2) \in R \end{cases}$$

There was considerable variation in the way people expressed manipulations of strings, ranging from the solution above to little bits of Clojure code involving `drop` and indexing into strings, like $p[0]$ and $p[1]$. I accepted all of those (if they were otherwise correct), even when they seemed to confuse strings and sets (assuming, again, that the meaning was otherwise clear and correct).

Languages

Consider the lower-case alphabet $A = \{"a", ..., "z"\}$ and the set
$C = A \cup \{"(", ")", "\neg", " \vee ", " \wedge "\}$ of characters.
We define a small language $\mathcal{L} \subseteq C^*$ of propositional formulae over the set of variable names $V = A^* \setminus \{\varepsilon\}$, and the following set of rules $R = \{R_1, R_2, R_3\}$ with

$R_1 = \{(s, "\neg" s) : s \in C^*\}$
$R_2 = \{(s_1, s_2, "(" s_1 " \vee " s_2 ")") : s_1, s_2 \in C^*\}$
$R_3 = \{(s_1, s_2, "(" s_1 " \wedge " s_2 ")") : s_1, s_2 \in C^*\}$

such that $\mathcal{L} = R[V]$.

1. [1 p] Show that $\mathcal{L} \subset C^*$ by giving a string $s \in C^*$ such that $s \notin \mathcal{L}$:
   $s = ($
   Hint: Make sure the strings are in $C^* \setminus \mathcal{L}$!
2. [3 p] Give three strings $s_1, s_2, s_3 \in C^* \setminus \mathcal{L}$ such that $(s_1, s_2, s_3) \in R_3$:
   $s_1 =)$
   $s_2 = ($
   $s_3 = () \wedge ()$ Note that $(s_1, s_2, s_3) \in R_3$ -- many solutions missed that point.
3. [7 p] Assume a function $E : V \longrightarrow \{0, 1\}$ that assigns every variable name a value in $\{0, 1\}$. Using **structural recursion**, define an evaluation function $\text{eval}_E : \mathcal{L} \longrightarrow \{0, 1\}$ that interprets the formulae in $\mathcal{L}$ in a way consistent with the usual interpretation of the symbols $\neg$, $\vee$, and $\wedge$ in propositional logic. **Use arithmetic operators (+, -, *, min, max) to compute with the values 0 and 1.**

$$\text{eval}_E : s \mapsto \begin{cases} E(s) & \text{for } s \in V \\ 1 - eval_E(s') & \text{for } s = \neg s' \\ max(eval_E(s_1), eval_E(s_2)) & \text{for } s = (s_1 \vee s_2) \\ min(eval_E(s_1), eval_E(s_2)) & \text{for } s = (s_1 \wedge s_2) \end{cases}$$

In many solutions the eval function could produce values outside of {0, 1} --- for example, if the second clause is simply -eval(s'), it may result in -1, and if the third is eval(s1) + eval(s2), you may get 2 etc.