The aim of the course is to grasp the mathematical definition of the meaning (or, as we say, the semantics) of programs in two paradigms: logic programming (a remarkable example is the Prolog programming language) and functional programming (like Haskell or Scala).

Perhaps surprisingly, the mathematical framework for both of these paradigms is more or less the same. Thus, the first part of the course (the one dealing with Logic Programming) is a bit more involved in math – in the second part (dealing with Functional Programming) we will be able to re-use most of the mathematic material covered in the first part.

Semantics of logic programs

As an introduction, we give some examples of logic programs and the intended semantics of them. A logic program is simply a (not necessarily finite!) set of program clauses. That is, there is no particular ordering of the clauses as in the case, say, imperative programs: a program here is just a set of constraints, a set of logical formulas describing the world in which the programming environment tries to derive facts, from a set of known facts, applying a set of inference rules.

A program clause is a formula of the form \( p_1 \land p_2 \land \ldots \land p_n \rightarrow q \), where each \( p_i \) and \( q \) are atomic formulas. For those readers not involved with logic: in a structure, each of these \( p_i \) and \( q \) evaluate to either 0 (false) or 1 (true); the conjunction \( p_1 \land p_2 \land \ldots \land p_n \) evaluates to 1 if and only if all the \( p_i \) are 1, that is, \( \land \) is the „minimum” operator[1] and the implication \( F \rightarrow G \) evaluates to 1 if and only if the value of \( F \) is at most the value of \( G \) (that is, if \( F \) is true, then \( G \) has to be true as well). Note that \( \land \) has a higher precedence than \( \rightarrow \): we have to evaluate the body \( p_1 \land \ldots \land p_n \) of the clause first, and then compare this value to the value of the head \( q \) of the clause.

So a rule, or a formula of the form \( p_1 \land \ldots \land p_n \rightarrow q \) basically states that “if all of the statements \( p_1, p_2, \ldots, p_n \) hold, then \( q \) holds as well”, and a program is simply a set of such rules.

It can happen that \( n = 0 \), that is, a clause can have an empty body which is written as \( \rightarrow q \). Note that the usage of the logical connectives and the direction of the implication is not consistent in the literature: there are people writing \( q \leftarrow p_1 \land \ldots \land p_n \), or even \( q \leftarrow p_1, \ldots, p_n \); in Prolog, these rules are written as \( q:-p_1, \ldots, p_n \) and when the head is empty, then it’s \( q \), ending with a period instead of the \( :\) sign but this is only a notational difference.

Now when the body is empty, that’s evaluated to 1 (for reasons becoming apparent later on), thus if \( \rightarrow q \) is true, then \( q \) is true as well. That’s why program clauses having an empty body are called facts and clauses having a nonempty body are called inference rules, usually.

Consider the following example for a first-order logic program consisting of three clauses.

\[
\begin{align*}
\rightarrow & \text{ even}(0) \\
\text{even}(x) & \rightarrow \text{ odd}(s(x)) \\
\text{odd}(x) & \rightarrow \text{ even}(s(x))
\end{align*}
\]

[1] later on, we will be more precise with that and it’ll be called the infimum of the values.
Given a program, a **structure** is always some object which assigns “meanings” to the elementary expressions present in the program; in the case of first-order logics, a structure consists of

- an **universe**, that is, some set $A$ of objects;
- to each $n$-ary function symbol (i.e. a function symbol taking $n$ inputs), there is an associated **function** from $A^n$ to $A$;
- to each $n$-ary predicate symbol there is an associated **predicate**, that is, a mapping from $A^n$ to $\{0, 1\}$, the set of truth values.

So the difference between **functions** and **predicates** is their output: each of these take $n$ objects as input, but while functions produce an object, predicates produce a truth value. For example, when the set of objects is the set $\mathbb{N} = \{0, 1, 2, \ldots\}$ of natural numbers (note: in this course, 0 counts as a natural number), then addition and multiplication are (binary) **functions** (mapping pairs of naturals to naturals), the **successor** function $n \mapsto n + 1$ is also a (unary) function, while equality, **less-than** and **is-even** are **predicates**: $= (n, m)$ holds iff $n = m$ (the equality relation has this meaning in every possible structure), $< (n, m)$ holds iff $n < m$ (now the symbol $<$ is not necessarily defined in an arbitrary set: e.g. if we choose the set of complex numbers as universe, then it’s not clear how should we interpret $<$), and **isEven**($n$) is true if and only if $n$ is an even number. So the first two are binary predicates, while the last one is a unary predicate.

It also makes sense to use **nullary** functions and predicates: a **nullary** function is simply a constant (mathematically, an $A^0 \rightarrow A$ function, but such functions can be identified with their unique image element), and a **nullary predicate** is a Boolean value. (This will be important later on.)

Then, in turn, in the example above, **even** and **odd** have to be (unary) predicate symbols (since their output is fed into an implication, so it has to be a Boolean value), while $s$ and $0$ have to be (unary and nullary, respectively) function symbols (since their output is fed to some predicate, thus it has to be an object as well). Also, $x$ is an object-valued variable.

Continuing our example program above, we can interpret our program in the **structure** where the **universe** is the set $\mathbb{N}$ of natural numbers; $s$ is interpreted by the successor function $n \mapsto n + 1$; **odd**($n$) holds for the number $n$ if $n$ is an odd number; and **even**($n$) holds for $n$ if $n$ is an even number. Also, let us interpret the constant 0 with the number 0. (Probably this is the structure the programmer had in mind; but the virtual machine interpreting the program has no clue about what the programmer had in mind, it can only see the function and the predicate symbols and make formal, symbolic computations using them.)

There are three rules stated that i) 0 is an even number; ii) if $x$ is an even number, then $x + 1$ is an odd number; and iii) if $x$ is an odd number, then $x + 1$ is an even number. (Note that the variables are implicitly quantified universally in a logic program.)

These statements hold in our structure, so this structure is a **model** of the program.

**Another structure** could be the one in which the universe is the set $\mathbb{Z}$ of the integers, 0 is interpreted by the number 0, $s$ is the negation $x \mapsto -x$, **even**($n$) holds if $n \geq 0$, and **odd**($n$) holds if $n \leq 0$. Then the clauses formalize the sentences i) 0 is a nonnegative number, ii) if $x$ is nonnegative, then $-x$ is not positive and iii) if $x$ is not positive, then $-x$ is nonnegative. All of the statements hold again in this structure, so this one is also a model of the program.

**Yet another structure** is the one in which the universe is the set $\mathbb{N}$ of the natural numbers,
\(s\) is the doubling function \(n \mapsto 2n\), **even** holds for even numbers, **odd** holds for the odd numbers. Then the meaning of the clauses is i) \(0\) is even (which is true), ii) if \(x\) is even, then \(2x\) is odd (which is false), and iii) if \(x\) is odd, then \(2x\) is even (which is true). Since the second clause is not satisfied, this one is *not* a model of the program.

As we can see, there can be many models of a program. The question of semantics is the following:

Among all the models of a program, which one should we choose?

The “chosen” model is called the **semantics** of a program.

Now the first transformation of a first-order logic program is called a **Herbrand extension**, which turns the program into a propositional logic program – albeit the size of the resulting program can be infinite (and it is infinite in most of the cases).

In order to define the Herbrand extension, we have to introduce the ground terms first:

**Definition: Ground term.**

The set of ground terms is the least set such that

- constant symbols are ground terms,
- if \(f\) is an \(n\)-ary function symbol and \(t_1, \ldots, t_n\) are ground terms, then \(f(t_1, \ldots, t_n)\) is a ground term.

Basically, any finite string that can be built up starting from constant symbols, applying function symbols (respecting their arity) is a ground term. For example, if \(f\) is a unary function symbol, \(g\) is a binary function symbol, and \(0\) is a constant, then \(0, f(0), f(f(0)), g(0, 0)\) and \(g(f(0), f(g(0, 0)))\) are ground terms.

**Definition: Herbrand extension.**

Given a first-order logic program \(P\), its **Herbrand extension** is the logic program we get by substituting ground terms in place of its variables in every possible way.

Continuing our running example, the ground terms are \(0, s(0), s(s(0))\), and so on, in general terms of the form \(S^n(0)\) (as \(0\) is the only constant and \(S\) is the only (unary) function symbol).

Then the Herbrand extension of our first-order program contains the clauses

\[
\begin{align*}
\rightarrow & \text{ even}(0) \\
\text{ even}(0) & \rightarrow \text{ odd}(s(0)) \\
\text{ odd}(0) & \rightarrow \text{ even}(s(0)) \\
\text{ even}(s(0)) & \rightarrow \text{ odd}(s(s(0))) \\
\text{ odd}(s(0)) & \rightarrow \text{ even}(s(s(0))) \\
\text{ even}(s(s(0))) & \rightarrow \text{ odd}(s(s(s(0)))) \\
& \ldots
\end{align*}
\]
The reason why the Herbrand extension is preferred over the original logic program is that the resulting clauses contain only ground formulas (that is, variable-free formulas) and thus the atomic parts (that begin with a predicate) can be viewed simply as Boolean variables.

That is, in the example above, \texttt{even}(0), \texttt{odd}(0), \texttt{even}(s(0)) and so on are actually Boolean variables. Then, a model of the above program becomes simply an assignment of Boolean variables, no fancy structures with some universe and interpretation for the function and predicate symbols are needed. Instead, there might be an infinite set of Boolean variables and an infinite number of clauses – which is still easier to manage in practice.

Also, the original program \( P \) and its Herbrand extension \( P' \) are tightly related: a model of \( P \) can be transformed into a model of \( P' \) and vice versa, so if we can choose a model for the Herbrand extension, we also choose a model for the original program as well.

Viewing the Herbrand extension of our running example, a possible model is the assignment which assigns 1 to the variables \texttt{even}(0), \texttt{odd}(s(0)), \texttt{even}(s(s(0))), \ldots, and 0 to the others, that is, \texttt{odd}(s^n(0)) is true if and only if \( n \) is odd and \texttt{even}(s^n(0)) is true if and only if \( n \) is even. (The reader is encouraged to check that this assignment indeed satisfies all the clauses above.)

This (Boolean) assignment corresponds actually to one of the two models of the original program we’ve already seen: to the one in which we set the universe as the natural numbers and interpret \( s \) with the successor function, and \texttt{even}, \texttt{odd} are interpreted by the corresponding parity checker predicates.

Also, if we set the universe to be a singleton \( \{0\} \), and interpret \( s(0) = 0, \texttt{even}(0) = \texttt{odd}(0) = 1 \) (that is, both predicates hold true for the single element of the universe), then it’s also a model of our first-order program; that model corresponds in the Herbrand extension to the satisfying assignment in which we set all the variables to true.

And again, there are many models of the transformed program.

It suffices to give a semantics for propositional logic programs (containing possibly an infinite number of clauses and variables) – first-order programs will get their semantics from that via the Herbrand extension.

Also, it’s worth observing that the assignment which sets every variable to 1 is always a satisfying assignment (since if the head of a clause is true, then the clause is satisfied), but it’s probably not the one the programmer had in mind.

So our primary aim is the following:

Given a propositional logic program, that is, a (possibly infinite) set \( P \) of clauses of the form

\[
p_1 \land p_2 \land \ldots \land p_n \rightarrow q,
\]

where each \( p_i \) and \( q \) are Boolean variables from a (possibly infinite) set \( Z \), give a “good” model of \( P \) as “the” semantics of \( P \).

Now we’ll see that in more complicated cases (in particular, in the case of generalized logic programs) it’s not always clear what makes a model “good”...
A “good” semantics minimizes the truth values.

But in order to understand this sentence, we’ll need to mathematically define what’s exactly “minimized” here.

In general, we’ll work with partially ordered sets, or posets for short:

**Definition: Poset.**

A relation $\leq$ over a set $P$ is called a partial order on $P$ if it satisfies all the following conditions:

- $x \leq x$ for each $x \in P$ (reflexivity);
- $x \leq y$ and $y \leq z$ imply $x \leq z$ for each $x, y, z \in P$ (transitivity);
- if $x \leq y$ and $y \leq x$ hold for $x, y \in P$, then $x = y$ (antisymmetry).

In this case $(P, \leq)$ is called a partially ordered set, or simply a poset.

If the poset additionally satisfies that for each $x, y \in P$ we either have $x \leq y$ or $y \leq x$ (that’s called dichotomy), then $(P, \leq)$ is a linearly ordered set.

For examples: the set $\mathbb{N}$ of naturals with their standard ordering $\leq$ is a linearly ordered set, and so are the sets $\mathbb{Z}$ of integers, $\mathbb{Q}$ of rationals and $\mathbb{R}$ of reals.

When $X$ is a set, then $P(X)$ is the power set of $X$, which consists of the subsets of $X$; then, $(P(X), \subseteq)$ is a poset which is not linearly ordered (apart from the cases when $|X| \leq 1$). For example, we can depict the “Hasse diagram” $P(X)$ with $X = \{p, q, r\}$ as

```
{p,q,r}
/\1
{p,q}{p,r}{q,r}
/\/\/
{p} {q} {r}
/\1/
\0
```

So, $P(X)$ has eight elements in this case. In a Hasse diagram, $x \leq y$ holds if and only if $y$ can be reached from $x$ via some “elevating” path in the diagram. (Of course there might be problems with this interpretation if the poset is infinite as it’s frequently the case.) Clearly, this $P(X)$ is not linearly ordered, since e.g. $\{p\}$ and $\{q\}$ (and many other pairs) are incomparable elements of the poset, neither of them being a subset of the other one.

Another type of posets is the poset denoted $X_\perp$ where $X$ is some set: in this poset, the underlying set is $X \cup \{\perp\}$ for the new element $\perp$, and the ordering is that $\perp \leq x$ for every member $x$ of the poset, and all the other elements are incomparable. For example, with $X = \{1, 2, 3\}$, the poset $X_\perp$ is

```
1 2 3
\/\/
\perp
```
which is also not a linearly ordered poset (again, apart from the cases when $|X| \leq 1$).

We will frequently use two particular posets: the first of them is the poset $2$ of the Boolean values, with $\{0, 1\}$ as the set and $0 \leq 1$ as the ordering, that is,

$$
\begin{array}{c}
1 \\
0
\end{array}
$$

Clearly, $2$ is a linearly ordered set (being essentially the same as – isomorphic to – $P(\{1\})$, the power set of a singleton set and also as $\{1\}_\perp$, the pointed poset of a singleton set).

The other poset will be the poset of assignments. Let $Z$ be the set of (Boolean) variables (once and for all – note that it’s a set, without any particular ordering!) Then, an assignment is a function $u : Z \to 2$ – a mapping from the set of variables to the set $\{0, 1\}$. In general, when $X$ and $Y$ are sets, then $X^Y$ denotes the set of $Y \to X$ functions, thus $2^Z$ stands for the set of the assignments.

Since $2$ is a poset, we can turn $2^Z$ into a poset as well, with the pointwise ordering. In general, when $P$ is a poset and $I$ is a set, then $P^I$ (again: this is the set of functions $I \to P$) is a poset with the ordering $u \leq v$ if and only if $u(i) \leq v(i)$ for all $i \in I$. That is, if $u$ and $v$ are functions, then we say $u \leq v$ if the value $u(i)$ (which is an element of the poset $P$) is at most the value $v(i)$ for all possible inputs $i \in I$.

Suppose $Z = \{p, q, r\}$. Then the assignments in $2^Z$ can be represented as vectors of length three: $(x, y, z)$ represents the assignment where the value of $p$ is $x$, the value of $q$ is $y$ and the value of $r$ is $z$.

Then, $(0, 0, 1) \leq (1, 0, 1)$ since $\leq$ holds on all three coordinates; but for example, $(0, 1, 0)$ and $(1, 0, 1)$ are incomparable elements (since the first one is greater on the second coordinate, while the second one is greater on the first and the third coordinate). Hence the poset $P^I$ is usually not linearly ordered, even if $P$ is.

In the case $Z = \{p, q, r\}$, the Hasse diagram of the poset $2^Z$ is

```
(1,1,1)
(1,1,0)     (1,0,1)     (0,1,1)
|    |       |       |
(1,0,0)   (0,1,0)   (0,0,1)
|    |       |       |
(0,0,0)
```

which is the same as $P(Z)$! This is true in general: the posets $2^Z$ and $P(Z)$ are always isomorphic under the mapping $u \mapsto \{p \in Z : u(p) = 1\}$. We will stick to the notation $2^Z$ since at some point later we’ll use logics with three or four possible truth values and it will be more convenient to use notations like $3^Z$ instead of “hacking” three possible values into the lattice of $P(Z)$.

So, $2^Z$ is a poset with the pointwise ordering.
We also need the following definitions in order to understand the part of “minimizing” the truth values:

**Definition: Minimal and least elements.**

If $P$ is a poset and $X \subseteq P$ is a subset of the poset, then $x \in X$ is...

- a **minimal** element of $X$ if $\forall y \in X \ y \leq x \Rightarrow y = x$; (there is no element of $X$ which is strictly less than $x$)
- the **least** element of $X$ if $\forall y \in X \ x \leq y$; ($x$ is less than all the other elements of $X$)

Dually, $x \in X$ is...

- a **maximal** element of $X$ if $\forall y \in X \ x \leq y \Rightarrow y = x$; (there is no element of $X$ which is strictly greater than $x$)
- the **largest** element of $X$ if $\forall y \in X \ y \leq x$; ($x$ is larger than all the other elements of $X$)

Note that in a subset $X$ of $P$ there can be many minimal elements. For example, if we have the formula $F = p \lor q \lor r$, then its models are $(1,0,0)$, $(0,1,0)$, ..., $(1,1,1)$, basically every member of $2^{\{p,q,r\}}$ is a model of $F$ but $(0,0,0)$. Then, the set of the models of $F$ has **three minimal elements**, $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$ (since there is no model of $F$ which is strictly less than any of them), but there is no **least model** of $F$. Of course it can also happen that a set $X$ has no minimal elements at all (e.g. when our poset is $\mathbb{Z}$ and $X$ is the set of all the negative numbers).

But the **least element is always unique** (meaning if it exists, then there is only one):

**Proposition**

If $P$ is a poset, $X \subseteq P$ and $x,y$ are least elements of $X$, then $x = y$.

**Proof**

Since $x$ is a least element of $X$ and $y \in X$, we get $x \leq y$. Similarly, since $y$ is a least element of $X$ and $x \in X$, we get $y \leq x$. Applying antisymmetry we get $x = y$.

Also, if there is a least element $x$, then $x$ is the only minimal element.

So the “minimizing the truth value” part can be formalized as

A “good” semantics of $P$ is an assignment which is a **minimal** model (according to the pointwise ordering on $2^P$).

So we should construct an assignment $u$ which satisfies all clauses of $P$ and there is no other model $v \neq u$ of $P$ with $v \leq u$. This is what we mean by “minimizing truth values”. Hence, for our example formula $F = p \lor q \lor r$ (which is not a logic program, only a formula) a good semantics would be one of $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$ but not the other four models.
The function $T_P$

On possible approach to seek for a minimal model is the following:

1. Start from an initial assignment, say $(0,0,\ldots,0)$, setting all the variables to 0.
2. Evaluate all the bodies.
3. In the next iteration, set a variable $q$ to 1 if and only if there is a clause which has $q$ as its head and whose body is evaluated to 1 according to our current iteration.
4. Repeat Steps 2–3 till all the clauses are satisfied.

As an example, when we have the following program

\[
\begin{align*}
&\rightarrow p & p \rightarrow q & p \rightarrow r & p \land q \rightarrow s \\
&t \rightarrow s & t \rightarrow q & p \land t \rightarrow s
\end{align*}
\]

then in the first iteration (let us fix the order $(p,q,r,s,t)$ in assignment) we start from the assignment $(0,0,0,0,0)$. Then we evaluate all the bodies: only the first rule $\rightarrow p$ has a body with value 1 (since empty bodies always evaluate to 1), all the others are false. Since the head of $\rightarrow p$ is $p$, $p$ is set to 1 in the next iteration; all the other variables remain 0. Then our new assignment is $(1,0,0,0,0)$.

In the next step, the clauses $\rightarrow p$, $p \rightarrow q$ and $p \rightarrow r$ have bodies evaluating to 1, thus their heads, $p,q$ and $r$ are set to 1. Our new assignment $(1,1,1,0,0)$.

In the next step, $\rightarrow p$, $p \rightarrow q$, $p \rightarrow r$ and $p \land q \rightarrow s$ have bodies evaluating to 1, thus their heads, $p,q,r$ and $s$ are set to 1. Our new assignment is $(1,1,1,1,0)$.

In the next step, $t \rightarrow s$, $t \rightarrow q$ and $p \land t \rightarrow s$ still have bodies evaluating to 0, our new assignment is still $(1,1,1,1,0)$.

Actually, $(1,1,1,1,0)$ is the least model of the program in the example, thus it’s the only minimal model, meaning that this is the only possible semantics the program can have.

What we have just done: we iterated some function which took some assignment and produced some other assignment. That is, a function from $2^\mathbb{Z} \rightarrow 2^\mathbb{Z}$.

Formalizing this function $T_P$ associated to the program $P$ we get the following:

**Definition: The function $T_P$.**

\[
T_P(u)(q) := \bigvee_{p_1 \land p_2 \land \ldots \land p_n \rightarrow q \in P} u(p_1) \land u(p_2) \land \ldots \land u(p_n).
\]

That is, if $u \in 2^\mathbb{Z}$ is the current assignment, then $T_P(u)$ is the new assignment; the formula says that the value of a variable $q$ in the new assignment should be calculated as follows: first we collect all the clauses in $P$ having $q$ as head, then we evaluate their bodies (that’s the $u(p_1) \land \ldots \land u(p_n)$ part), and then take the disjunction of the values – if there is at least one evaluated to 1 according to the current assignment, then the new value of $q$ will be 1, otherwise it’s set to 0.

Szabolcs Iván, University of Szeged, Hungary 8
Suprema and infima

In the definition of the function $T_P$ we used the symbols $\lor$ and $\land$ – the question is, how should we interpret these operations when there is e.g. an infinite number of values inside the $\lor$ (that is, when $q$ appears as the head of infinitely many clauses)? Or the case when we have an empty conjunction (when the body is empty) or an empty disjunction (when there are no clauses having the head $q$)?

To give a mathematically supported answer to these questions, we recall the following notions:

**Definition: Lower and upper bounds, infima and suprema.**

When $P$ is a poset and $X \subseteq P$, then an element $y \in P$ is...

- an upper bound of $X$, denoted $X \leq y$, if $\forall x \in X \ x \leq y$;
- the supremum of $X$, denoted $y = \lor X$, if it is the least upper bound of $X$;
- a lower bound of $X$, denoted $y \leq X$, if $\forall x \in X \ y \leq x$;
- the infimum of $X$, denoted $y = \land X$, if it is the greatest lower bound of $X$.

It can happen for a set $X \subseteq P$ that $X$ has absolutely no lower or upper bounds, and also that $X$ does have upper bounds but there is no least upper bound etc.

Consider the pointed poset $\{1, 2, 3\}_\bot$. There, the set $\{1, 2\}$ has the lower bound $\bot$, which is its infimum as well, but there is no upper bound of this set (and thus it has no supremum).

For another example, considering the poset $\mathbb{Q}$ of rationals, equipped with their standard ordering, and setting $X = \{x \in \mathbb{Q} : x \leq \sqrt{2}\}$ as the set of those rationals being smaller than $\sqrt{2}$ (which is known to be an irrational number but it still can be compared to rationals of course), then $X$ has infinitely many upper bounds (any rational number larger than $\sqrt{2}$ is an upper bound) but there is no least of them (among the rationals of course).

Also, if we take the poset $\mathbb{N}$ with the standard ordering, then any nonempty subset $X \subseteq \mathbb{N}$ has a least element, so nonempty subsets have infima. Also, finite sets have suprema (namely, their largest element), but infinite subsets of $\mathbb{N}$ have no upper bounds at all. If we enrich $\mathbb{N}$ with the additional element $\infty$ and setting $n \leq \infty$ for each $n \in \mathbb{N}$, then the resulting poset $\mathbb{N} \cup \{\infty\}$ is so that every subset has a supremum (finite nonempty subsets’ suprema is their largest element; infinite subsets’ suprema is $\infty$ and for $\emptyset$, we will get back to it in a moment).

It is worth to study the case $X = \emptyset$ separately. First, it holds for any $y \in P$ that $\emptyset \leq y$ since the claim $\forall x \in X \ x \leq y$ is vacuously satisfied. Hence every member of the poset $P$ is an upper bound of $\emptyset$, thus the least upper bound has to be the least element of $P$.

That is, $\lor \emptyset$ is always the least element of the poset, which is usually denoted $\bot$; if there is no least element of $P$, then $\lor \emptyset$ does not exist. Similarly, $\land \emptyset$ has to be the largest element of $P$.

Thus, the $\lor$ and $\land$ operations on $2$ are actually the supremum and infimum operators: $\lor\{0\} = \lor\emptyset = 0$ and $\lor\{1\} = \lor\{0, 1\} = 1$ and similarly, $\land\{0\} = \land\emptyset = \land\{0, 1\} = 0$ and $\land\{1\} = \land\emptyset = 1$. This last one explains why should we evaluate empty bodies to 1: an empty conjunction has to have...
the value 1, and an empty disjunction has to have the value 0. Also, the definition also makes it clear that an “infinitary disjunction” should be handled as follows: first, we collect all the values appearing in the disjunction into a single set, and then we take the supremum of this given set.

We write \( X \leq Y \) if \( \forall x \in X \forall y \in Y \ x \leq y \), that is, every member of \( Y \) is an upper bound of every member of \( X \).

The following fact is easy to check:

**Proposition**

Assume \( X \leq Y \) for the subsets \( X, Y \) of \( P \).

i) If \( \bigwedge Y \) exists, then \( X \leq \bigwedge Y \leq Y \).

ii) If \( \bigvee X \) exists, then \( X \leq \bigvee X \leq Y \).

iii) If both \( \bigwedge Y \) and \( \bigvee X \) exist, then \( X \leq \bigvee X \leq \bigwedge Y \leq Y \).

**Proof**

It is clear that \( X \leq \bigvee X \) since \( \bigvee X \) is an upper bound (namely, the least one) of \( X \) by definition and similarly for \( \bigwedge Y \leq Y \).

Now let \( X \leq Y \) and assume \( \bigvee X \) exists. Then for each \( y \in Y \), we have \( X \leq y \), that is, \( y \) is an upper bound of \( X \). Since \( \bigvee X \) is the least upper bound of \( X \), we get \( \bigvee X \leq y \) for every \( y \in Y \), which is the same as writing \( \bigvee X \leq Y \).

Similarly, if \( \bigwedge Y \) exists, then each \( x \in X \) is a lower bound of \( Y \), thus \( x \leq \bigwedge Y \), implying \( X \leq \bigwedge Y \).

Now if both \( \bigvee X \) and \( \bigwedge Y \) exist, then we already know \( \bigvee X \leq Y \). Applying i) with \( \{ \bigvee X \} \) playing the role of \( X \) there we get \( \bigvee X \leq \bigwedge Y \leq Y \).

It is also clear that if one drops several non-maximal elements of a set (in such a way there is at least a retained upper bound of any dropped element), then the set of upper bounds do not change:

**Proposition**

Suppose \( X, Y \) are subsets of the poset \( P \) such that for any element \( x \in X \) there exists an element \( y \in Y \) with \( x \leq y \).

Then any upper bound of \( Y \) is also an upper bound of \( X \).

**Proof**

Assume \( Y \leq z \). We have to show that \( X \leq z \). For this, let \( x \in X \) be a member of \( X \). By the condition on \( X \) and \( Y \), there is an element \( y \in Y \) with \( x \leq y \). Since \( Y \leq z \), we also have \( y \leq z \), and by transitivity we get \( x \leq z \).

Thus, in particular, if \( P \) has the least element \( \bot \), and \( X \subseteq P \), then the set of upper bounds of \( X \) and \( X \cup \{ \bot \} \) coincide, and thus if \( \bigvee X \) exists, then \( \bigvee X = \bigvee (X \cup \{ \bot \}) \). Indeed: since to
each $x \in X$ there is the same $x$ in $X \cup \{\bot\}$, this shows any upper bound of $X \cup \{\bot\}$ is also an upper bound of $X$; and to each $x \in X \cup \{\bot\}$, either $x \in X$ or $x = \bot$, and in the latter case there is an upper bound of $x$ in $X$, if $X$ is nonempty. Finally, if $X = \emptyset$, then $P$ is the set of upper bounds of $\emptyset$ and $\{\bot\}$ as well.

Another handy fact on taking suprema is that suprema can be rearranged:

**Proposition**

Let $P$ be a poset, $I$ and $J$ be index sets, and to each $i \in I$, $j \in J$ let $x_{i,j} \in P$ be an element. Then

$$\bigvee_{i \in I} \bigvee_{j \in J} x_{i,j} = \bigvee_{i \in I, j \in J} x_{i,j},$$

provided all suprema on the left hand side exist.

**Proof**

Suppose $y$ is an upper bound of the set $\{x_{i,j} : i \in I, j \in J\}$. Then for each $i \in I$, $y$ is an upper bound of $\{x_{i,j} : j \in J\}$ (since this latter set is a subset of the previous one). Since $\bigvee_{j \in J} x_{i,j}$ is the least upper bound of this set, we get $\bigvee_{j \in J} x_{i,j} \leq y$ for each $i \in I$. Hence $y$ is an upper bound of these suprema and hence $y$ is also an upper bound of their supremum, that is, $\bigvee_{i \in I} \bigvee_{j \in J} x_{i,j} \leq y$. Thus, $\bigvee_{i \in I} \bigvee_{j \in J} x_{i,j}$ is a lower bound for each upper bound $y$ of $\{x_{i,j} : i \in I, j \in J\}$, hence in particular

$$\bigvee_{i \in I, j \in J} x_{i,j} \leq \bigvee_{i \in I, j \in J} x_{i,j}.$$

For the other direction, suppose $y$ is an upper bound of the suprema $\bigvee_{j \in J} x_{i,j}$ for each $i \in I$. Then for each $i \in I$ and $j' \in J$ we have $x_{i,j'} \leq \bigvee_{j \in J} x_{i,j}$ since $x_{i,j'}$ is a member of the set whose supremum is taken on the right side. Thus, $x_{i,j'} \leq y$ as well (since $y$ is an upper bound of $\bigvee_{j \in J} x_{i,j}$) for each $i \in I$, $j' \in J$, implying $y$ is an upper bound of $\{x_{i,j} : i \in I, j \in J\}$ as well. Hence, since $\bigvee_{i \in I} \bigvee_{j \in J} x_{i,j}$ is the least upper bound, we get by choosing $y = \bigvee_{i \in I, j \in J} x_{i,j}$ that

$$\bigvee_{i \in I, j \in J} x_{i,j} \leq \bigvee_{i \in I} \bigvee_{j \in J} x_{i,j},$$

thus the two values indeed coincide.

Thus in particular, $\bigvee_{i \in I} \bigvee_{j \in J} x_{i,j} = \bigvee_{j \in J} \bigvee_{i \in I} x_{i,j}$ if all the suprema on both sides exist.

**Complete lattices**

As we have seen, suprema do not necessarily exist. But when they do, that’s a “nicely ordered” poset deserving a name:

**Definition: Complete lattice.**

A poset $P$ is called a **complete lattice** if every subset of $P$ has a supremum.
For example, $2$ is a complete lattice (since we already enumerated all four subsets of $2$ and calculated the supremum of each one).

In particular, every complete lattice has a least element, usually denoted $\bot$, since $\bigvee \emptyset$ also has to exist since $\emptyset$ is a subset of the poset, and we already argued that $\bigvee \emptyset$ is always the least element of the poset.

A nice property of complete lattices is that infima also exist:

**Proposition**

If $P$ is a complete lattice, then each $X \subseteq P$ also has an infimum as well.

**Proof**

Let $X \subseteq P$ be a subset of the poset. Let $Y = \{ y \in P : y \leq X \}$ be the set of the lower bounds of $X$. Then $Y \leq X$ and since $P$ is a complete lattice, $\bigvee Y$ exists. Thus, $Y \leq \bigvee Y \leq X$. We claim that $\bigvee Y = \bigwedge X$.

Indeed, since $\bigvee Y \leq X$, we have that $\bigvee Y$ is a lower bound of $X$. But then, $\bigvee Y \in Y$ since $Y$ contains all the lower bounds of $X$. Thus, since $\bigvee Y$ is an upper bound for $Y$, we have that $y \leq \bigvee Y$ for each $y \in Y$, along with $\bigvee Y \in Y$ we get that $\bigvee Y$ is the greatest element of $Y$, that is, the greatest lower bound of $X$, thus $\bigvee Y = \bigwedge X$.

Of course, $\bigvee P$ is the greatest element of $P$, so a complete lattice always has a greatest element which is usually denoted $\top$.

Now since $2$ is a complete lattice, the right-hand side of the function $T_P$ is well-defined for any program $P$ since it consists of evaluations, mapping $2^Z$ to $2$, then taking infima and suprema within $2$, which always exist.

Next, we show that $2^Z$, the poset of assignments, is also a complete lattice. We do it a bit more general way:

**Proposition**

If $P$ is a complete lattice, then so is $P^I$ for any index set $I$.

In particular, suprema are to taken pointwise: if $U \subseteq P^I$ is a set (of functions from $I$ to $P$), then their supremum is the function $\bigvee U : I \rightarrow P$ with $(\bigvee U)(i) = \bigvee_{u \in U} u(i)$ for each $i \in I$.

**Proof**

Let $U$ be a subset of $P^I$. Since $P$ is a complete lattice, the suprema $\bigvee_{u \in U} u(i)$ indeed exist for each $i \in I$, since $\{u(i) : u \in U\}$ is a subset of $P$ which always have a supremum in a complete lattice. So let $u^*$ be the function defined as above: $u^*(i) := \bigvee_{u \in U} u(i)$ for all $i \in I$.

We claim that $u^*$ is indeed the least upper bound of $U$.

First we show that $u^*$ is an upper bound. Let $u \in U$, we have to show that $u \leq u^*$. Since $P^I$ is equipped with the pointwise ordering, this is equivalent to $u(i) \leq u^*(i)$ for each $i \in I$. But this is clear since $u^*(i)$ is the suprema of the set $\{v(i) : v \in U\}$ and $u(i)$ is a member of this set since $u \in U$. Thus $u^*$ is indeed an upper bound.

Next we show that if $v$ is an upper bound of $U$, then $u^* \leq v$. That is, we have to show...
$u^*(i) \leq v(i)$ for each $i$. Since $U \leq v$, we get that $u \leq v$ for each $u \in U$, yielding $u(i) \leq v(i)$ for each $u \in U$ and $i \in I$.

This means that $v(i)$ is an upper bound of $\{u(i) : u \in U\}$ and since $u^*(i)$ is the least upper bound of this set, we get $u^*(i) \leq v(i)$ for each $i \in I$, hence $u^* \leq v$. Thus $u^*$ is indeed the supremum of $U$.

Thus, $2^Z$ is also a complete lattice. (And since $2^Z$ is isomorphic to $P(Z)$, so is each poset of the form $P(Z)$.)

**Models of $P$ are the pre-fixed points of $T_P$**

The reason why we study the function $T_P$ is its intimate relation to the set of models of the program $P$:

**Proposition**
The assignment $u \in 2^Z$ is a model of $P$ if and only if $T_P(u) \leq u$.

**Proof**

Let $u \in 2^Z$ be an assignment.

Then, $u$ is not a model of $P$ if and only if there is a clause $p_1 \land \ldots \land p_n \rightarrow q \in P$ which is false under $u$.

This is further equivalent to stating that there is a clause $p_1 \land \ldots \land p_n \rightarrow q \in P$ with $u(p_1) \land \ldots \land u(p_n) = 1$ and $u(q) = 0$.

This is further equivalent to stating that there is a variable $q$ and a clause $p_1 \land \ldots \land p_n \rightarrow q \in P$ with $u(p_1) \land \ldots \land u(p_n) = 1$ and $u(q) = 0$.

This is further equivalent to stating that there is a variable $q$ such that $\bigvee_{p_1 \land \ldots \land p_n \rightarrow q} u(p_1) \land \ldots \land u(p_n) = 1$ and $u(q) = 0$, since this supremum is 1 if and only if there is a clause whose body evaluates to 1.

By the definition of $T_P$, this is further equivalent to stating that there exists a variable $q$ such that $T_P(u)(q) = 1$ and $u(q) = 0$. Since in $2$ there are only these two possible truth values, this is equivalent to stating that $T_P(u)(q) \not\leq u(q)$ for some variable $q$, which means exactly that $T_P(u) \not\leq u$, as needed.

The property “$f(x) \leq x$” for some function $f$ is again a nice property deserving a name:

**Definition: Pre-fixed, post-fixed and fixed points of a function.**

When $P$ is a poset and $f : P \rightarrow P$ is a function, then $x \in P$ is...

- a pre-fixed point of $f$ if $f(x) \leq x$;
- a post-fixed point of $f$ if $x \leq f(x)$;
• a fixed point of $f$ if $x = f(x)$.

So once again, we can reformulate our aim:

If $\mathcal{P}$ is a logic program, then its semantics should be a minimal pre-fixed point of the function $T_\mathcal{P}$.

Indeed: we know that the semantics should be a minimal model of $\mathcal{P}$, and models of $\mathcal{P}$ are precisely the pre-fixed points of $T_\mathcal{P}$.

In the following, we will show several smaller facts, which together give us a unique semantics for a logic program:

1. We will show that $T_\mathcal{P}$ is a so-called “continuous” function.
2. We will show that whenever $f : \mathcal{P} \to \mathcal{P}$ is a continuous function, with $\mathcal{P}$ being a complete lattice, then $f$ has a least pre-fixed point.

These two statements together with $2^\mathcal{Z}$ being a complete lattice give us the answer, namely: the semantics of $\mathcal{P}$ has to be this least pre-fixed point of $T_\mathcal{P}$, since if there is a least pre-fixed point, then it is the only minimal one.

Monotone and continuous functions

There are two important properties of functions, monotonicity and continuity, which give us methods to calculate (pre-)fixed points.

**Definition: Monotonicity.**

A function $f : \mathcal{P} \to \mathcal{Q}$ from a poset $\mathcal{P}$ into a poset $\mathcal{Q}$ is monotone if $x \leq y$ implies $f(x) \leq f(y)$.

The definition of continuity is somewhat more involved. Recall that from calculus, a function $f : \mathbb{R} \to \mathbb{R}$ is called continuous, if the image of some limit is the same as the limit of the images, i.e., $\lim_{n \to \infty} f(x_n) = f(\lim_{n \to \infty} x_n)$, provided the sequence $x_1, x_2, \ldots$ is convergent. Our definition of continuity is very similar to that one: in this field, suprema play the role of limits, so image of the supremum should be the same of the supremum of the images. Instead “convergence” we have “if the supremum exists”. Also, the set of real numbers forms a linear order, which is not necessarily true for a poset. Thus, we include an additional assumption, requiring the “sequence” (rather, a set) to be linearly ordered.

The definition is the following:

**Definition: Continuity.**

A function $f : \mathcal{P} \to \mathcal{Q}$ from a poset $\mathcal{P}$ into a poset $\mathcal{Q}$ is continuous if whenever $X \subseteq \mathcal{P}$ is so that

- $X$ is nonempty,
- $X$ is linearly ordered,
- and $\bigvee X$ exists,
then $f(\bigvee X) = \bigvee_{x \in X} f(x)$.

We will shortly see that continuity implies monotonicity. The following observation comes
handy in proving this (and a couple of things later):

**Proposition**

When $P$ is a poset and $x, y \in P$, then $x \leq y$ if and only if $y = \bigvee \{x, y\}$.

**Proof**

Assume $x \leq y$. Then by $y \leq y$ we have that $y$ is an upper bound of $\{x, y\}$. It is also the
least upper bound since if $z$ is also an upper bound in $\{x, y\}$, then in particular $y \leq z$,
thus $y$ is a lower bound of any upper bound. Thus, $y = \bigvee \{x, y\}$.

For the reverse direction, if $y = \bigvee \{x, y\}$, then $y$ is an upper bound of $\{x, y\}$, in particular
$x \leq y$.

Now we are ready to show that continuity is stronger:

**Proposition**

If a function $f : P \to Q$ is continuous, then it is also monotone.

**Proof**

Assume $f : P \to Q$ is continuous and let $x \leq y$ be elements of $P$. We have to show that
$f(x) \leq f(y)$. Now if $x \leq y$, then $\{x, y\}$ is a nonempty (it has either one or two elements,
one iff $x = y$), linearly ordered (by $x \leq y$, each pair of elements are comparable of $\{x, y\}$)
subset of $P$, and its supremum exists: $\bigvee \{x, y\} = y$. Then, since $f$ is continous,

$$f(y) = f(\bigvee \{x, y\}) = \bigvee \{f(x), f(y)\};$$

so $f(y)$ is an upper bound of the set $\{f(x), f(y)\}$, implying $f(x) \leq f(y)$.

**The Tarski Fixed Point Theorem**

Having a monotone function $f : P \to P$ is good for various reasons: one of them is that the
image of a pre- or post-fixed point remains a pre- or post-fixed point:

**Proposition**

Assume $f : P \to P$ is a monotone function. If $x$ is a pre- or post-fixed point of $f$, then so
is $f(x)$.

**Proof**

From $x \leq f(x)$ we get that (applying monotonicity on both sides) $f(x) \leq f(f(x))$, that
is, if $x$ is a post-fixed point, then so is $f(x)$.

Similarly, if $x$ is a pre-fixed point, then $f(x) \leq x$, which in turn implies by monotonicity
that \( f(f(x)) \leq f(x) \), thus \( f(x) \) is a pre-fixed point as well.

Given a poset \( P \) having a least element \( \bot \), it is always a post-fixed point, since \( \bot \leq f(\bot) \), whatever \( f(\bot) \) is (since \( \bot \) is the least element). Applying \( f \) on both sides we get \( f(\bot) \leq f^2(\bot) \), then again applying \( f \) we have \( f^2(\bot) \leq f^3(\bot) \) and so on (more formally: from \( f^n(\bot) \leq f^{n+1}(\bot) \) we get \( f^{n+1}(\bot) \leq f^{n+2}(\bot) \) by one application of \( f \), and since the statement holds for \( n = 0 \), we get by induction that it holds for all integers \( n \geq 0 \).

Thus when \( f : P \to P \) is monotone for the poset \( P \) having the least element \( \bot \), we have

\[
\bot \leq f(\bot) \leq f^2(\bot) \leq f^3(\bot) \leq \ldots
\]

(which is usually called an \( \omega \)-chain).

We claim that the supremum of this chain is the least pre-fixed point of \( f \), moreover, it is even a fixed point:

**Proposition: Tarski Fixed Point Theorem.**

Suppose \( P \) is a poset having the least element \( \bot \), \( f : P \to P \) is a continuous function and the supremum \( x^* \) of the set \( \{f^n(\bot) : n \geq 0\} \) exists.

Then \( x^* \) is the least pre-fixed point of \( f \), and moreover, it is also a fixed point. (So it is the least fixed point as well, since fixed points are pre-fixed points themselves.)

**Proof**

First we show that the \( x^* \) above is a lower bound for any pre-fixed point. So let \( x \) be a pre-fixed point of \( f \). We show that \( \{f^n(\bot) : n \geq 0\} \leq x \).

To show this, we have to prove that \( f^n(\bot) \leq x \) for each \( n \geq 0 \), which can be done via induction on \( n \). For the base case \( n = 0 \) the statement holds since \( f^0(\bot) = \bot \), and \( \bot \leq x \), whatever \( x \) is.

Assume the claim holds for \( n \): \( f^n(\bot) \leq x \). Since \( f \) is continuous, it is also monotone. Hence we can apply \( f \) on both sides and get \( f^{n+1}(\bot) \leq f(x) \). But since \( x \) is a pre-fixed point, we also have \( f(x) \leq x \), thus \( f^{n+1}(\bot) \leq x \) also holds, which proves that \( x \) is an upper bound of \( \{f^n(\bot) : n \geq 0\} \). Since \( x^* \) is the least upper bound of this set, we get that \( x^* \leq x \), i.e., \( x^* \) is a lower bound of any pre-fixed point.

Next we show that \( x^* \) is a fixed point of \( f \). We have already seen that \( \{f^n(\bot) : n \geq 0\} \) is a linearly ordered (and of course nonempty) set, moreover, its supremum (that is, \( x^* \)) exists. Thus, applying continuity of \( f \) we get

\[
f(x^*) = f(\bigvee \{f^n(\bot) : n \geq 0\}) = \bigvee_{n \geq 0} f(f^n(\bot)) = \bigvee_{n \geq 0} f^{n+1}(\bot)
\]

\[
= \bigvee \{f(\bot), f^2(\bot), \ldots\} = \bigvee \{\bot, f(\bot), f^2(\bot), \ldots\} = x^*,
\]

so \( x^* \) is indeed a fixed point.

(Note that in the last step we used the fact \( \bigvee X = \bigvee (X \cup \{\bot\}) \) we have seen earlier.)
Hence, if \( f : P \to P \) is a continuous function with \( P \) being a complete lattice (in which case the supremum above exists) then the least (pre)fixed point of \( f \) can be “computed” via a fixed-point iteration: we start from the element \( \perp \), and repeatedly apply \( f \) on the result of the previous step, finally we take the supremum of all these values (technically, this “computation” may not terminate since we have to produce the infinite sequence \( \perp, f(\perp), f^2(\perp), \ldots \) first).

We have seen that the least pre-fixed point of the above \( f \) is a fixed point as well. This is true in a more general setting:

**Proposition**

Assume \( f : P \to P \) is a monotone function and \( x \) is a minimal pre-fixed point of \( f \). Then \( x \) is a fixed point of \( f \).

**Proof**

Since \( x \) is a pre-fixed point, we have \( f(x) \leq x \). Since \( f \) is monotone, applying \( f \) on both sides we get \( f(f(x)) \leq f(x) \). Thus, \( f(x) \) is also a pre-fixed point of \( f \), moreover, \( f(x) \leq x \). Since \( x \) is assumed to be a minimal pre-fixed point of \( f \), it has to be the case \( f(x) = x \), that is, \( x \) is a fixed point.

**\( T_P \) is continuous**

In this part we show that the function \( T_P \) defined earlier is a continuous function. We’ll break the proof in several parts.

**Definition: Projections.**

When \( P \) is a poset, \( I \) is some set, and \( i \in I \), then the \( i \)th projection from \( P^I \) to \( P \) is the function \( \pi_i : P^I \to P \) defined as

\[
\pi_i(u) := u(i).
\]

That is, we evaluate the \( i \)th coordinate of the function \( u \).

For example, \( \pi_2(x, y, z) = y \), \( \pi_1(1, 0, 1) = 1 \) and \( \pi_q(u) = u(q) \), when \( q \in \mathbb{Z} \) and \( u \in \mathbb{2}^\mathbb{Z} \). That is, if \( u \) is a variable assignment, then a projection is simply the value of a single variable according to the given assignment. Such guys \( u(p_i) \) are building block of the definition of \( T_P \). And they are continuous:

**Proposition**

Projections are continuous.

**Proof**

Let \( P \) be a poset, \( I \) be a set and \( i \in I \). We want to show that \( \pi_i : P^I \to P \) is continuous. To this end, let \( U \subseteq P^I \) be a nonempty, linearly ordered set of functions whose supremum \( u^* = \bigvee U \) exists. We have to show \( \pi_i(u^*) = \bigvee_{u \in U} \pi_i(u) \).

But this is clear, since we already know that in \( P^I \), suprema are to be taken pointwise,
i.e., \[ \pi_i(u^*) = u^*(i) = (\bigsqcup U)(i) = \bigvee_{u \in U} u(i) = \bigvee_{u \in U} \pi_i(u). \]

The next construct used when building up \( T_P \) is called a target tupling. Basically, if we have a set of functions \( f_i : P \to Q, \ i \in I \) for some index set \( I \), then we can make one single function of them, which outputs the “tuple” or “vector” containing all the results of the \( f_i \) functions:

**Definition: Target tupling.**

If \( P \) and \( Q \) are posets, \( I \) is some set and to each \( i \in I, f_i : P \to Q \) is a function from \( P \) to \( Q \), then their target tupling is the function \( \langle f_i \rangle_{i \in I} : P \to Q^I \) defined as
\[
\langle f_i \rangle_{i \in I}(x)(i) = f_i(x)
\]
for each \( x \in P \) and \( i \in I \).

For example, when we take the (binary) conjunction and disjunction \( \land, \lor : 2^2 \to 2 \), then their target tupling in this order is the function \( \langle \land, \lor \rangle \) is a function from \( 2^2 \) into \( 2^2 \), defined as
\[
\langle \land, \lor \rangle(x, y) = (x \land y, x \lor y).
\]
(Note that this function sorts its input as \((0,0) \rightarrow (0,0), (1,0) \rightarrow (0,1), (0,1) \rightarrow (0,1) \) and \((1,1) \rightarrow (1,1) \).)

Whenever we have a function \( f : P \to Q^n \) for some integer \( n \geq 0 \) (that is, a function that outputs an \( n \)-ary vector), it can always be written as \( f = \langle f_1, \ldots, f_n \rangle \), the target tupling of \( n \) functions, each \( f_i \) being a function from \( P \) to \( Q \): \( f_1 \) computes the first coordinate of the output, \( f_2 \) computes the second, and so on. As in the previous example, the result of sorting two bits is a pair of bits, the first is the smaller value (that’s the infimum, computed by \( \land \)), the second is the greater value (computed by \( \lor \)).

When we evaluate the body of a clause, first we compute the values \( u(p_1), u(p_2), \ldots, u(p_n) \), then (say) arrange them to a vector of the form \( (u(p_1), \ldots, u(p_n)) \) (that’s done by target tupling of the projections), and after that we apply the \( n \)-ary conjunction function \( \land_n : 2^n \to 2 \).

In the next step we show that the function \( u \mapsto (u(p_1), \ldots, u(p_n)) \) is continuous:

**Proposition**
The target tupling of continuous functions is continuous.

**Proof**

Let \( f_i : P \to Q, i \in I \) be continuous functions and let \( f : P \to Q^I \) stand for their target tupling \( \langle f_i \rangle_{i \in I} \).

Let \( X \subseteq P \) so that \( X \) is nonempty, linearly ordered and has the supremum \( x^* = \bigvee X \). We have to show that \( f(x^*) = \bigvee_{x \in X} f(x) \). Since \( f \) is a function from \( P \) to \( Q^I \), these values are functions from \( I \) to \( Q \); thus the above equality holds if and only if \( f(x^*)(i) = \bigvee_{x \in X} f(x)(i) \).

From the definition of target tupling we have \( f(x^*)(i) = f_i(x^*) \), which further equals to \( f_i(\bigvee X) = \bigvee_{x \in X} f_i(x) \) since \( f_i \) is continuous. Writing back the definition of target tupling we get this is \( \bigvee_{x \in X} (f(x)(i)) \) which is the same as \( \bigvee_{x \in X} f(x)(i) \), since in the poset \( Q^I \)
Now we know that the function $u$ defined as $(x_1, \ldots, x_n) \mapsto x_1 \wedge \ldots \wedge x_n$ is also continuous:

**Proposition**

The function $\wedge_n : 2^n \to 2$ is continuous for any $n \geq 0$.

**Proof**

Let $X \subseteq 2^n$ be a nonempty, linearly ordered subset of $2^n$ having the supremum $x^*$. We have to show $\wedge_n(x^*) = \bigvee_{x \in X} \wedge_n(x)$.

Since $2^n$ itself is finite, $X$ is finite as well. Thus, it’s a finite, nonempty, linearly ordered set, hence it can be written as $X = \{x_1, \ldots, x_k\}$ with $x_1 \leq x_2 \leq \ldots \leq x_k$. In particular, it has a greatest element $x_k$, thus $x^* = x_k$.

Since $\wedge_n$ maps into $2$, we only have to show that $\wedge_n(x_k) = 1$ if and only if $\bigvee_{x \in X} \wedge_n(x) = 1$. But that’s clear since $\wedge_n(x_k) = 1$ iff $x_k = (1, 1, \ldots, 1)$. Since $x_k$ is the largest element of $X$, and the largest element of $2^n$ as well, this is further equivalent to $(1, 1, \ldots, 1) \in X$, which in turn is equivalent to $\bigvee_{x \in X} \wedge_n(x) = 1$.

Now we know that the function $u \mapsto (u(p_1), \ldots, u(p_n))$ is continuous, and so is $(u(p_1), \ldots, u(p_n)) \mapsto u(p_1) \wedge \ldots \wedge u(p_n)$. The function $u \mapsto u(p_1) \wedge \ldots \wedge u(p_n)$ is the composition of these functions which also preserves continuity:

**Proposition**

Composition of continuous functions is also continuous.

That is, if $f : P \to Q$ and $g : Q \to R$ are continuous functions, then so is $g \circ f : P \to R$ defined as $(g \circ f)(x) = g(f(x))$.

**Proof**

Let $X \subseteq P$ be a nonempty, linearly ordered subset of $P$, having the supremum $x^*$. We have to show that $(g \circ f)(x^*) = \bigvee_{x \in X} (g \circ f)(x)$.

First, let us observe the subset $f(X) = \{f(x) : x \in X\}$ of $Q$. Since $X$ is nonempty, $f(X)$ is nonempty as well. Also, if $y_1, y_2 \in f(X)$, then $y_1 = f(x_1)$ for some $x_1$ and $y_2 = f(x_2)$ for some $x_2$. Since $f$ is monotone, $x_1 \leq x_2$ implies $y_1 = f(x_1) \leq f(x_2) = y_2$, and similarly, $x_2 \leq x_1$ implies $y_2 \leq y_1$. Thus, since each $x_1, x_2 \in X$ are comparable in $P$ (because $X$ is linearly ordered), each $y_1, y_2 \in f(X)$ are also comparable in $Q$. Thus, $f(X)$ is also a nonempty, linearly ordered subset of $Q$.

Now we can proceed as

\[
(g \circ f)(x^*) = g(f(\bigvee_{x \in X} f(x))) \quad \text{now applying continuity of } f
\]

\[
= g(\bigvee_{x \in X} f(x)) \quad \text{now since } \{f(x) : x \in X\} \text{ is nonempty,}
\]

\[
\text{linearly ordered, has a supremum, and } g \text{ is continuous,}
\]

\[
= \bigvee_{x \in X} g(f(x)),
\]
which is exactly we need.

Thus, we have that the functions of the form \( u \mapsto u(p_1) \land \ldots \land u(p_n) \) are continuous. In the definition of \( T_P \), the supremum of such functions is taken. This, again, preserves continuity:

**Proposition**

If \( I \) is a set and to each \( i \in I \), \( f_i : P \to Q \) is a function, then their supremum \( \bigvee_{i \in I} f_i : P \to Q \), defined as

\[
\left( \bigvee_{i \in I} f_i \right)(x) = \bigvee_{i \in I} (f_i(x)),
\]

if exists, it is continuous.

**Proof**

Note that the supremum of functions does not always exist. For example, when \( P = 2 \), and \( Q \) is the pointed poset \( \{1, 2\}_\perp \), then if \( f_1(0) = 1 \) and \( f_2(0) = 2 \), then \( (f_1 \lor f_2)(0) \) should be the supremum of \( \{f_1(0), f_2(0)\} \), which is \( \{1, 2\} \) but in \( \{1, 2\}_\perp \), that supremum does not exist.

But in the current case, when \( Q \) is a complete lattice, the supremum always exists.

So assume \( X \subseteq P \) is a nonempty, linearly ordered set having the supremum \( x^* \). We have to show \( (\bigvee_{i \in I} f_i)(x^*) = \bigvee_{x \in X} (\bigvee_{i \in I} f_i)(x) \). Applying the continuity of each \( f_i \) we get

\[
\left( \bigvee_{i \in I} f_i \right)(x^*) = \bigvee_{i \in I} \left( \bigvee_{x \in X} f_i(x) \right)
\]

and

\[
\bigvee_{x \in X} \left( \bigvee_{i \in I} f_i \right)(x) = \bigvee_{x \in X} \left( \bigvee_{i \in I} f_i(x) \right)
\]

by the definition of the supremum function, and these two values coincide as we have seen already.

So far we have shown that all the functions of the form

\[
u \mapsto \bigvee_{p_1 \land \ldots \land p_n \rightarrow q \in \mathcal{P}} u(p_1) \land \ldots \land u(p_n)
\]

are continuous. These functions map from \( 2^Z \to 2 \). The function \( T_P : 2^Z \to 2^Z \) is actually the **target tupling** of such functions! To each variable \( q \in Z \) we have a function of the above form, and we arrange the results into a “tuple” indexed by \( Z \) (that is, we get a function \( Z \to 2 \), an assignment). Since we have already seen that the target tupling of continuous functions is continuous, we get that \( T_P \) is continuous as well.

**Summary: Logic Programs**

1. A logic program is a set \( \mathcal{P} \) of clauses of the form \( p_1 \land \ldots \land p_n \rightarrow q \) with each \( p_i \) and \( q \) being Boolean variables drawn from a set \( Z \). Both \( Z \) and the number of clauses in \( \mathcal{P} \) can be infinite.
2. A model of the program $P$ is an assignment $u : Z \to 2$, where $2$ is the set $\{0, 1\}$ with the ordering $0 \leq 1$ of truth values, which satisfies all the clauses of $P$. The set of assignments is also denoted $2^Z$.

3. We associated a function $T_P$ to a program $P$, which transforms assignments into assignments: the new value of a variable $q$ is 1 if and only if there exists a clause whose body is 1 according to the old assignment, and whose head is $q$.

4. We showed that $T_P$ is a “continuous function”.

5. We showed that $2^Z$ is a “complete lattice”.

6. We showed that the models of $P$ are exactly the “pre-fixed points” of $T_P$.

7. We argued that we should seek for a “minimal” model of $P$.

8. We proved the Tarski Fixed Point Theorem which states that there is exactly one minimal pre-fixed point of a continuous function $f$ on a complete lattice; this minimal pre-fixed point is actually a least pre-fixed point, moreover, it is also a fixed point as well and can be constructed as the supremum of the sequence $\bot, f(\bot), f^2(\bot), \ldots$, where $\bot$ denotes the least element of the lattice.

9. Thus, there is only one choice for a “good” semantics of a program $P$: namely, we start from the all-zero assignment $\bot$, iterate $T_P$ and take the supremum of the resulting sequence.

Summarizing the math results and introducing a name for this semantics,

To a logic program $P$, we associate the following function $T_P : 2^Z \to 2^Z$:

$$T_P(u)(q) = \bigvee_{p_1 \land \ldots \land p_n \rightarrow q \in P} u(p_1) \land \ldots \land u(p_n).$$

Then, the canonical semantics of $P$ is

$$\bigvee_{n \geq 0} T_P^n(\bot)$$

with $\bot$ being the all-zero assignment.

The canonical semantics is the least model of $P$. Moreover, it is additionally a fixed point of $T_P$. By the way, fixed points of $T_P$ are called supported models of $P$. 
Generalized Logic Programs

In this part we extend the framework developed in the previous part to generalized logic programs. Such a program is a (possibly infinite) set of clauses of the form

\[ p_1 \land p_2 \land \ldots \land p_n \land \neg q_1 \land \neg q_2 \land \ldots \land \neg q_k \rightarrow r, \]

where each \( p_i, q_j \) and \( r \) are variables, again drawn from a (possibly infinite) set \( Z \). That is, negated variables can appear in the body of a clause but the head is always a (positive) variable.

We want to retain most parts of the previous framework. So, we can define the function \( T_P : 2^Z \rightarrow 2^Z \) similarly to the previous case:

\[ T_P(u)(r) = \bigvee_{p_1 \land \ldots \land p_n \land \neg q_1 \land \ldots \land \neg q_k \rightarrow r \in P} u(p_1) \land \ldots \land u(p_n) \land \neg u(q_1) \land \ldots \land \neg u(q_k). \]

Again, models of \( P \) are precisely the pre-fixed points of \( T_P \).

However, there are serious problems with continuity and even with monotonicity.

Consider the example program

\[-p \land \neg q \rightarrow r
\neg q \land \neg r \rightarrow p
\neg p \land \neg r \rightarrow q\]

Then, if we start from the assignment \((0, 0, 0)\) (the ordering of the variables is \( p, q, r \) as in the previous examples), then every body gets evaluated to 1 (as \( \neg 0 \land \neg 0 \) is 1), thus the new value of all the variables is set to 1, that is, \( T_P(0, 0, 0) = (1, 1, 1) \) which is a model of \( P \), so far so good.

But then, iterating \( T_P \) once more, all the bodies are evaluated to 0 this time, thus each variable is set to 0 again. That is, \( T_P(1, 1, 1) = (0, 0, 0) \) which is not that good: this \( T_P \) is not a monotone function!

Another problem we face is that this program, viewed as the conjunction of its clauses, is equivalent to the formula \( p \lor q \lor r \) – thus it has three minimal models as we have seen in one of our starting examples, there is no least model. However, in this case these three minimal models, \((0, 0, 1), (0, 1, 0)\) and \((1, 0, 0)\) are all fixed points of \( T_P \), so they are supported models of \( P \).

But, considering the even smaller program \( \neg p \rightarrow p \), in that case \( T_P(0) = 1 \) and \( T_P(1) = 0 \), thus the only model \( p = 1 \) is not a fixed point, it’s not a supported model.

So the main problems are: \( T_P \) is not always monotone; it does not always have a least pre-fixed point; it does not always have a fixed point at all.

The 4-valued logic

There are more options to resolve these issues. We choose the following path:
Instead of the logical values 0 and 1, we assign intervals of truth values to the variables.

In general, when $P$ is a poset, $P^2$ can be seen as the set of intervals of $P$, with $(x, y)$ representing the set $\{ z : x \leq z \leq y \}$ of elements of $P$ between the two endpoints.\footnote{Note that $x$ and $y$ are members of $(x, y)$ in this formalism. In calculus, one writes $[x, y]$ for intervals like these.}

That is,

**Definition: Values of the 4-valued logic.**

Elements of $4 = 2 \times 2$ are denoted as follows:

- The element $(0, 0)$ represents the set $\{0\}$ “can be only false”, and is denoted $\mathbf{f}$.
- The element $(1, 1)$ represents the set $\{1\}$ “can be only true”, and is denoted $\mathbf{t}$.
- The element $(0, 1)$ represents the set $\{0, 1\}$ “unknown, can be both”, and is denoted $\perp$.
- The element $(1, 0)$ represents the empty set, “inconsistent, cannot be assigned”, and is denoted $\top$.

The first three elements are called the consistent elements of $4$.

In general, an element $(x, y)$ of $P^2$ for an arbitrary poset $P$ is called consistent if $x \leq y$.

We define two partial orders: the truth order $\leq_t$ and the precision order $\leq_p$ on $P^2$:

**Definition: Truth order and precision order.**

For a poset $P$ we define the following partial orders $\leq_t$ and $\leq_p$ on $P^2$:

$$(x, y) \leq_t (x', y') \iff x \leq x' \text{ and } y \leq y'.$$

$$(x, y) \leq_p (x', y') \iff x \leq x' \text{ and } y' \leq y.$$

Basically, if $x \leq y$ denotes in $P$ that $y$ is “more true” than $x$, then $(x, y) \leq_t (x', y')$ denotes (more or less) that the interval $(x', y')$ is more true than the interval $(x, y)$. For the order $\leq_p$, $(x, y) \leq_p (x', y')$ holds if the interval $(x', y')$ is contained inside the interval $(x, y)$, thus, in a sense, $(x', y')$ is a “more precise” interval than $(x, y)$.

We will want to take suprema and infima of subsets of $P^2$ with respect to both orderings, hence we have to use different notations for these operations in order to be distinguishable.

**Definition: Infima and suprema in $P^2$.**

In $P^2$,

- $\bigvee$ denotes the supremum operation with respect to $\leq_t$;
- $\bigwedge$ denotes the infimum operation with respect to $\leq_t$;
- $\bigoplus$ denotes the supremum operation with respect to $\leq_p$;
• \( \otimes \) denotes the infimum operation with respect to \( \leq_p \).

The following is clear\(^3\).

**Proposition**

When \( P \) is a poset and \( X = \{(x_i, y_i) : i \in I\} \) is a subset of \( P^2 \), then

- \( \bigvee X = (\vee_i x_i, \vee_i y_i) \),
- \( \bigwedge X = (\wedge_i x_i, \wedge_i y_i) \),
- \( \bigoplus X = (\vee_i x_i, \wedge_i y_i) \) and
- \( \bigotimes X = (\wedge_i x_i, \vee_i y_i) \).

**Proof**

The poset \((P^2, \leq_t)\) is simply \( P^2 \) with the pointwise ordering, proving the first two items. For the third item, \((x^*, y^*)\) is an upper bound of \( X \) if \((x_i, y_i) \leq_p (x^*, y^*)\) for each \( i \in I \), that is, \( x_i \leq x^* \) and \( y^* \leq y_i \) for each \( i \in I \). That is, if and only if \( x^* \) is an upper bound of the \( x_i \) and \( y^* \) is a lower bound of the \( y_i \). Hence \((\vee_i x_i, \wedge_i y_i)\) is an upper bound for \( X \). Also, if \((x, y)\) is an upper bound of \( X \), then \( \{x_i : i \in I\} \leq x \), implying \( \vee_i x_i \leq x \) and similarly, \( y \leq \wedge_i y_i \) also holds, thus \((\vee_i x_i, \wedge_i y_i)\) is indeed the supremum with respect to \( \leq_p \). The fourth item can be proven analogously.

Thus, if \( P \) is a complete lattice, then so are \((P^2, \leq_t)\) and \((P^2, \leq_p)\) (since in a complete lattice all infima also exist, thus the right-hand sides of the previous Proposition are always defined). That’s why structures of the form \((P^2, \leq_t, \leq_p)\) are called bi-lattices.

Let us visualize the two orderings in \( 4 \) (using the aliases we introduced earlier: \( \top \) for \((1, 0)\), \( f \) for \((0, 0)\) etc):

\[
\begin{array}{c}
\top \\
\downarrow \\
f \\
\downarrow \\
\perp \leq_t \\
\end{array} \quad \begin{array}{c}
\top \\
\downarrow \\
f \\
\downarrow \\
\perp \leq_p \\
\end{array}
\]

The ordering \( \leq_p \) makes clear why we use \( \top \) for \((1, 0)\) and \( \perp \) for \((0, 0)\): these are the greatest/least elements of \( \leq_p \), respectively. Also, we use \( \wedge \) and \( \vee \) for infimum and supremum with respect to \( \leq_t \) since this makes \( t \wedge f = f \) and so on, so the “usual” semantics of \( \vee \) and \( \wedge \) are retained for these “point-like” intervals.

We already have the operations \( \vee \) and \( \wedge \) within \( 4 \), now we’ll define negation on intervals. But how should we do that? First, we should have \( \neg f = t \) and \( \neg t = f \) since we want to

---

\(^3\)It would be even more clear if we introduced the product posets \( \prod P_i \) and not only the special case \( P^I \). We will see how it goes this way.
extend the negation from 2 to 4. It also makes sense to define $\neg \bot = \bot$, since if we do not know anything about a variable’s value (i.e. it can be either 0 or 1), then we do not know anything about its negation. Similarly, it also makes sense to define $\neg \top = \top$, since if a value is contradictionary, then so is its negation. It can be checked that the following definition accomplishes this:

**Definition: Negation on 4.**
\[
\neg (x, y) = (\neg y, \neg x).
\]

Then, we can define the following function $\Phi_P : 4^Z \rightarrow 4^Z$ as follows:

**Definition: The function $\Phi_P$.**
\[
\Phi_P(u)(r) = \bigvee_{p_1 \wedge \ldots \wedge p_n \wedge \lnot q_1 \wedge \ldots \wedge \lnot q_k \rightarrow r \in P} u(p_1) \wedge \ldots \wedge u(p_n) \wedge \lnot u(q_1) \wedge \ldots \wedge \lnot u(q_k).
\]
That is, syntactically the above function coincides with $T_P$ defined earlier. The difference is the domain: while $T_P$ is a function over classical (binary) truth values, $\Phi_P$ works with assignments that assign intervals to each variable.

We have not defined the “truth table” for implication – it makes sense to define the value of $x \rightarrow y$ as $(-x) \lor y$, the latter two operations being already defined for intervals. Or, it also makes sense to set $x \rightarrow y$ to $t$ if $x \leq_t y$ and $f$ otherwise – both variants extend the classical case. If we choose to do the latter, then again, pre-fixed points of $\Phi_P$, with respect to the truth ordering $\leq_t$, are exactly the models of $P$.

**The current plan**

We outline the steps we will make in order to have a semantics for generalized logic programs, with assignments coming from $4^Z$:

1. We will transform $\Phi_P$ to an equivalent form $\Psi_P$, which will compute the very same function but which is easier to handle mathematically.

2. In particular, we will show that $\Psi_P$ is a monotone function with respect to $\leq_p$. It will not be continuous, though.

3. We will prove the Kleene Fixed Point Theorem, stating that any monotone function $P \rightarrow P$ has a least (pre)fixed point when $P$ is a complete lattice. Moreover, this least fixed point can be defined via some kind of fixed point iteration.

Then, we will call this least fixed point of $\Psi_P$ the Kripke-Kleene semantics of the program $P$. After that,

1. We will prove that the Kripke-Kleene semantics never has inconsistent values, so it’s essentially a 3-valued model.

2. We will outline some problems with the Kripke-Kleene semantics: most notably, it minimizes only with respect to $\leq_p$ but not with respect to $\leq_t$, which contradicts to the “a good semantics minimizes the truth values” rule.
3. We will introduce so-called stabilizer functions. Using these, we get from \( \Psi_P \) an “even better” function, also having a least fixed point with respect to \( \leq_p \).

4. We will show that this least fixed point of the stabilizer function is also a fixed point of \( \Psi_P \) which is also \( \leq_{r}\)-minimal.

We will call this least fixed point the well-founded semantics of \( P \), closing the part on Logic Programming.

The function \( \Psi_P \)

In this section we convert our function \( \Phi_P \), which is a \( 4^Z \rightarrow 4^Z \) to another function \( \Psi_P \), a \( 2^Z \times 2^Z \rightarrow 2^Z \times 2^Z \) function. The intuition is the following: \( \Phi_P \) gets as input two separate assignments, \( u \) and \( v \), both coming from \( 2^Z \). These two assignment together determine an interval-valued assignment in \( 4^Z \) in the following way: for a variable \( q \in Z \), \( u(q) \) gives the “left end-point” of the interval, and \( v(q) \) gives the “right end-point” of the interval.

Also, the output value of \( \Psi_P(u,v) \) will be a pair \((u',v')\) with \( u' \) and \( v' \) being assignments, coming from \( 2^Z \): \( u' \) will be the assignment computing the new left end-points, and \( v' \) will be the assignment computing the new right end-points.

Clearly, the function \( \Psi_P \) is basically the same as \( \Phi_P \), only the domain is transformed into an isomorphic one, from \((2 \times 2)^Z\) to \(2^Z \times 2^Z\). But again, the difference is only that while in \( \Phi_P \), the variables directly get an interval as value, in \( \Psi_P \) these intervals are decomposed to the two end-points.

Since the \( \Psi_P \) we want to construct is a function from \( 2^Z \times 2^Z \) to \( 2^Z \times 2^Z \), in particular, the output of the function is a pair of assignments, \( \Psi_P \) is the target tupling of two functions: let us call the function computing the new left end-points \( f_P \), and the function computing the new right end-points \( g_P \). Then, \( f_P \) and \( g_P \) are \( 2^Z \times 2^Z \rightarrow 2^Z \) functions.

Observing the function

\[
\Phi_P(u)(r) = \bigvee_{p_1, \ldots, p_n, \neg q_1, \ldots, \neg q_k \rightarrow r \in P} u(p_1) \land \ldots \land u(p_n) \land \neg u(q_1) \land \ldots \land \neg u(q_k),
\]

how is the new left end-point of the interval assigned to \( r \) calculated? It’s the left end-point of the interval

\[
\bigvee_{p_1, \ldots, p_n, \neg q_1, \ldots, \neg q_k \rightarrow r \in P} u(p_1) \land \ldots \land u(p_n) \land \neg u(q_1) \land \ldots \land \neg u(q_k).
\]

Since in \( P^2 \) we have seen that \( \bigvee_{i \in I} (x_i, y_i) = \bigvee_{i \in I} x_i \), we have to compute the left end-points of the intervals 

\[u(p_1) \land \ldots \land u(p_n) \land \neg u(q_1) \land \ldots \land \neg u(q_k)\]

and take their supremum. Also, the left end-point of \( \land \) of intervals is the \( \land \) of the left end-points of the same intervals, so we have to take the set of left end-points of the intervals 

\[u(p_1), \ldots, u(p_n), \neg u(q_1), \ldots, \neg u(q_k)\]

and take their infimum. Now \( \Psi_P \) takes as input two functions (assignments from \( 2^Z \)): \( u_1 \) and \( u_2 \) such that \( u(p) = (u_1(p), u_2(p)) \) for each \( p \in Z \). Thus, the left end-point of \( u(p_i) \) is \( u_1(p_i) \) for each \( 1 = 1, \ldots, n \). What’s the situation with the intervals of the form \( \neg u(q_j) \)? Well, since
It turns out that it’s enough to show
by the isomorphism between
4
ordering
≤
(Ψ
Our current aim is to show that Ψ
P
which is again a nice enough property deserving a name:
Szabolcs Iván, University of Szeged, Hungary 27 2016/10/07/23:26:35
Thus, the function Ψ
coordinates also get swapped, then the function is called symmetric.
That is, if a function maps pairs into pairs, and swapping the input coordinates the output
Observing carefully the definitions of the functions f
Symmetric and approximation functions
A symmetric function
A function f = ⟨f₁, f₂⟩: P × P → P × P is symmetric if f₁(x, y) = f₂(y, x) for each x, y ∈ P.
That is, if a function maps pairs into pairs, and swapping the input coordinates the output coordinates also get swapped, then the function is called symmetric.
Thus, the function Ψ_P is symmetric.
Our current aim is to show that Ψ_P is also ≤_p-monotone. That is, if (u, v) ≤_p (u′, v′), then
Ψ_P(u, v) ≤_p Ψ_P(u′, v′). (Note that this ≤_p is the precision ordering on 2^Z × 2^Z, that is, (u, v) ≤_p (u′, v′) iff (u(q), v(q)) ≤_p (u′(q), v′(q)) for each q ∈ Z. Technically, the precision ordering ≤_p is defined on 4 = 2 × 2, then it’s taken pointwise on 4^Z, finally it’s transformed by the isomorphism between 4^Z and 2^Z × 2^Z.)
It turns out that it’s enough to show ≤_p-monotonicity of f_P since:
Proposition
A symmetric function f = ⟨f₁, f₂⟩: P^2 → P^2 is ≤_p-monotone if and only if so is f₁.
Approximating means,

**Proposition**

If the function $f : P^2 \to P^2$ approximates the function $g : P \to P$, then $f(x, x) = (g(x), g(x))$ for each $x \in P$.

In more general, if $y \leq x \leq z$ (that is, $x$ is contained within the interval $(y, z)$), then $g(x)$ is contained in $f(y, z)$.

**Proof**

Clearly, $f(x, x) = (f_1(x, x), f_2(x, x)) = (f_1(x, x), f_1(x, x)) = (g(x), g(x))$, this holds for any symmetric function.

For the other statement, $y \leq x \leq z$ implies $(y, z) \leq_p (x, x)$, by $\leq_p$-monotonicity we get
\( f(y, z) \leq_p f(x, x) = (g(x), g(x)), \) that is, the point-like interval \((g(x), g(x))\) is contained in \(f(y, z)\).

Thus, approximation functions map more precise inputs to more precise outputs (that’s what \(\leq_p\)-monotonicity is stating) and output point-like intervals if the input is point-like (that’s implied by symmetry).

We already know that \(\Psi_P = \langle f_P, g_P \rangle\) is a symmetric function. We have not shown yet it’s also \(\leq_p\)-monotone, but nevertheless, we can ask the following: if it’s an approximation function, then what function does it approximate? Well, recalling that \(f_P(u_1, u_2)(r) = \bigvee_{p_1 \land \ldots \land p_n \land \neg q_1 \land \ldots \land \neg q_k \rightarrow r \in \mathcal{P}} u_1(p_1) \land \ldots \land u_1(p_n) \land \neg u_2(q_1) \land \ldots \land \neg u_2(q_k)\)

we get that \(\Psi_P\) then approximates the function \(u \mapsto f_P(u, u)\), that is,

\[ f_P(u, u)(r) = \bigvee_{p_1 \land \ldots \land p_n \land \neg q_1 \land \ldots \land \neg q_k \rightarrow r \in \mathcal{P}} u(p_1) \land \ldots \land u(p_n) \land \neg u(q_1) \land \ldots \land \neg u(q_k),\]

which is familiar... , yes, it’s the function \(T_P\) we started with!

So, if \(\Psi_P\) is an approximation function, then it approximates \(T_P\).

This “sounds good”, since \(T_P\) is deeply related to logic programs.

\(\Psi_P\) is \(\leq_p\)-monotone

The problem with \(T_P\) was that it’s not a monotone function, thus it offers no natural candidate for a semantics (i.e. a least pre-fixed point).

In this section we show that the function \(\Psi_P\) is monotone, with respect to the ordering \(\leq_p\).

Again, we will build up our proof bottom-up, breaking it into several smaller claims.

Also, we already know that it suffices to show that \(f_P\) is \(\leq_p\)-monotone.

We start with the literal evaluations again. In this case (since negation is involved) we only make the statement for the poset \(2^Z \times 2^Z\).

**Proposition**

The evaluation functions \((u_1, u_2) \mapsto u_1(p)\) and \((u_1, u_2) \mapsto \neg u_2(p)\) are \(\leq_p\)-monotone for each \(p \in \mathbb{Z}\).

**Proof**

Assume \((u_1, u_2) \leq_p (u_1', u_2')\), that is, \(u_1 \leq u_1' \) and \(u_2 \leq u_2'\) and let \(p \in \mathbb{Z}\). Then, since the \(u_i\) are pointwise ordered, we have \(u_1(p) \leq u_1'(p)\) and \(u_2'(p) \leq u_2(p)\). The former implies \((u_1, u_2) \mapsto u_1(p)\) is \(\leq_p\)-monotone. From the latter, \(u_2'(p) \leq u_2(p)\) implies \(\neg u_2(p) \leq \neg u_2'(p)\), hence \((u_1, u_2) \mapsto \neg u_2(p)\) is also \(\leq_p\)-monotone.

Then, we again build up \(f_P\) step by step the same way as before (during the proof of the continuity of \(T_P\)). But for monotonicity we don’t have to deal with finite infima separately:
Proposition

If $I$ is some index set and $f_i : P \to Q$ are monotone functions, then $\bigwedge_{i \in I} f_i$ and $\bigvee_{i \in I} f_i$, if they exist, are monotone as well.

Proof

Note that there is no need here to emphasize $\leq_p$-monotonicity: the argument works between arbitrary posets.

Let $x \leq y$. Then for each $i \in I$ we have $f_i(x) \leq f_i(y)$. Thus, any upper bound of $\{f_i(y) : i \in I\}$ is also an upper bound of $\{f_i(x) : i \in I\}$ (implying $\bigvee f_i(x) = \bigvee f_i(y)$) and similarly, any lower bound of $\{f_i(x) : i \in I\}$ is also a lower bound of $\{f_i(y) : i \in I\}$, implying $\bigwedge f_i(x) = \bigwedge f_i(y) = (\bigwedge f_i)(y)$.

Of course if $Q$ is a complete lattice, as in our case when $Q = 2^Z$, the supremum/infimum of such functions always exists.

Then, we have that the functions of the form

$$f_{P,r}(u_1, u_2) = \bigvee_{p_1 \land \ldots \land p_n \land \neg q_1 \land \ldots \land \neg q_k \rightarrow r \in P} u_1(p_1) \land \ldots \land u_1(p_n) \land \neg u_2(q_1) \land \ldots \land \neg u_2(q_k)$$

are $\leq_p$-monotone, since the literal evaluations are monotone, evaluating the body of the clause is then an infimum of $\leq_p$-monotone functions, which is then $\leq_p$-monotone as well, and then, taking the supremum of such functions is $\leq_p$-monotone again.

Then again, $f_P$ is the target tupling of such functions $f_{P,r}$, which also preserves monotonicity (and again, it does not matter that the ordering in question is $\leq_p$ or not):

Proposition

The target tupling of monotone functions is monotone.

Proof

Let $I$ be an index set and $f_i : P \to Q$ be monotone functions for each $i \in I$. We claim that the functions $f = \langle f_i \rangle_{i \in I} : P \to Q^I$ is monotone. So let $x \leq y$ be members of $P$, we have to show that $f(x) \leq f(y)$. Since $f(x)$ and $f(y)$ are from $Q^I$, i.e., functions ordered pointwise, $f(x) \leq f(y)$ if and only if $f(x)(i) \leq f(y)(i)$ holds for each $i \in I$. But that’s $f_i(x) \leq f_i(y)$ by the definition of target tupling, which holds since each $f_i$ is assumed to be monotone.

Thus,

Proposition

$\Psi_P$ is $\leq_p$-monotone.

Hence, it’s an approximation function since it’s symmetric as well.
Well-orderings and well-founded induction

Now we know that $Ψ_P$ is a monotone function (with respect to $≤_P$). Our current aim is to show that monotone functions always have least (pre)fixed points (in complete lattices, that is).

In order to achieve this, we have to meet a proof method called well-founded induction, which is a generalization of induction over the naturals we’ve already used (in the proof of the Tarski Fixed Point Theorem, which is not a coincidence since the Kleene Fixed Point Theorem generalizes the Tarski one, by generalizing the induction method).

But first, let us see an example for a monotone function in some complete lattice. Let the poset be $P = \mathbb{R}_{≥0} \cup \{∞\}$, the nonnegative real numbers equipped with a $∞$ element. We have seen that it’s a complete lattice.

For the function, we write each real number in the form $r = n - \alpha$, where $n$ is an integer and $0 < \alpha ≤ 1$. That is, $0.5 = 1 - 0.5$, $1.7 = 2 - 0.3$, $10.3 = 11 - 0.7$, and, for integers, $2 = 3 - 1$, $42 = 43 - 1$ and so on $n$ is the next strictly greater integer and $\alpha$ is the difference $n - r$. Then we define the function $f$ as

$$f(n - \alpha) = n - \frac{\alpha}{2} \quad f(∞) = ∞$$

For example, $f(0) = 0.5$, $f(0.5) = 0.75$, $f(42.2) = 42.6$ and so on.

Let’s try to produce a fixed point of $f$ by the iteration method we already seen in the proof of the Tarski Fixed Point Theorem: start from $0$, the least element of the poset, iterate $f$ and “at the end”, take the supremum:

$$x_0 = 0 \quad x_1 = f(0) = 0.5 \quad x_2 = f(0.5) = 0.75 \quad x_3 = f(0.75) = 0.875$$

and so on, in general $x_n = 1 - \frac{1}{2^n}$, thus the supremum of this sequence is $\bigvee_{n≥0} x_n = 1$.

If $f$ were continuous, we would have $f(1) = 1$ and we were done. However, $f(1) = 1.5$, so we are not done yet… for reasons becoming apparent later, let us refer to this element $\bigvee_{n≥0} x_n$ as $x_ω$. Then, iterating further we get

$$x_ω = 1 \quad x_ω+1 = f(1) = 1.5 \quad x_ω+2 = f(1.5) = 1.75 \quad x_ω+3 = f(1.75) = 1.875$$

and so on, and after another infinitely many iterations we take again a supremum and get a value, henceforth called $x_{ω×2} = 2$. Then again, $x_{ω×2+1} = 2.5$, $x_{ω×2+2} = 2.75$, and so on, iterating infinitely many times, taking supremum puts us into $x_{ω×3} = 3$ and so on.

And after infinitely many infinite iterations we arrive to $x_{ω×ω} = ∞$, so we “finally” reach the unique (pre)fixed point of $f$.

This might seem obscure at first but

it works

and it always works. Basically we just start from $⊥$, apply $f$ and take suprema “in some structured manner” and hack our way to the least (pre)fixed point of $f$.

Now for this “structured manner” we have to know a little more about well-orderings and well-founded induction.

\[^4\text{for those readers who have never done that before}^\]
The definition of well-ordering is as follows:

**Definition: Well-ordering.**

A strict linear order $(P, <)$ is a well-ordering if there is no infinite (strictly) descending chain

\[ \ldots < x_3 < x_2 < x_1 < x_0 \]

in $P$.

If we have a well-ordered set, then we are able to do well-founded induction:

**Proposition: Well-founded induction.**

Assume $(P, <)$ is a well-ordering and that $X \subseteq P$ is a set such that for any element $x \in P$, if all the elements $y < x$ strictly less than $x$ are in $X$, then so is $x$.

Then $X = P$.

Writing the statement in a bit more formal way:

\[ \forall x((\forall y(y < x \rightarrow x \in X)) \rightarrow x \in X) \]

implies $X = P$.

**Proof**

Assume $X$ satisfies the above property and $X \neq P$. Then there is some $x_0 \notin X$. Since $x_0$ is not in $X$, there has to be some element $x_1 < x_0$ also not in $X$ (since otherwise every element $y < x_0$ is in $X$, thus, by the condition, $x_0 \in X$ as well). Repeating this argument we get that there also exists some $x_2 < x_1$ not in $X$ and so on, and we get an infinite descending chain, which contradicts to $P$ being well-ordered.

In our example, we did not index our sequence $x_0, x_1, x_2, \ldots$ by only the natural numbers, but also by “something else” we called $\omega$, $\omega + 8$, $\omega \times 2$ and so on.

It turns out that we indexed our sequence with ordinals, another set theoretic construction. (We counted up to the ordinal $\omega^2$, which is still a fairly small ordinal – there are many more ordinals much larger than that.)

Ordinals have two nice properties why it is good to index sequences by them:

1. Any set of ordinals is well-ordered. (Thus, we can do well-founded induction on the sequence.)
2. It does not matter how large the cardinality of a set $X$ is, there is always an ordinal $\alpha$ which is “larger” than $X$ in the following sense: the set $\{\beta : \beta < \alpha\}$ of ordinals which are less than $\alpha$ has a larger cardinality than $X$. (Thus, we can count up to pretty much anything using ordinals. Even to uncountably infinite and more.)

**Ordinals**

In this section we define the (von Neumann) ordinals themselves. Basically, an ordinal is a set of sets, satisfying some additional properties (which ensure that the collection of ordinals is well-ordered).

\[ ^5\text{assuming ZFC, for those who are interested in the math details} \]
An ordinal is a set $\alpha$ of sets, satisfying the following properties:

- It is transitive: if $\beta \in \alpha$ and $\gamma \in \beta$, then $\gamma \in \alpha$ as well.
- The elements of $\alpha$ are well-ordered with respect to the $\in$ relation.

To put the second condition more explicit: if $x \in y$ and $y \in z$ for $x, y, z \in \alpha$, then $x \in z$ (transitivity); $x \notin x$ for each $x \in \alpha$ (irreflexivity); for each $x, y \in \alpha$, exactly one of $x \in y$, $y \in x$ or $x = y$ has to hold, moreover, there is no infinite sequence $x_1, x_2, \ldots \in \alpha$ such that $\ldots \in x_3 \in x_2 \in x_1$.

In particular, if $\alpha$ is an ordinal, then $\alpha \notin \alpha$, since if $\alpha \in \alpha$, then $\alpha$ (as an element of $\alpha$) violates the irreflexivity condition.

For example, the empty set $\emptyset$ is an ordinal (and is usually denoted 0). Also, $\{\emptyset\}$, the set containing the empty set is an ordinal: transitivity is OK, since $\alpha = \{\emptyset\}$, there is only one $\beta \in \alpha$, namely, $\beta = \emptyset$ and there is no $\gamma \in \emptyset$. Also, for $\{\emptyset\}$ being well-ordered it suffices to check irreflexivity, which holds since $\emptyset \notin \emptyset$. The ordinal $\{\emptyset\}$ is usually denoted 1.

Another ordinal is $\{\emptyset, \{\emptyset\}\}$ – that is, the two-element set, having the two previous ordinals 0 and 1 as elements. We could also write $\{0, 1\}$ for this ordinal, and call it 2. This ordinal 2 is transitive since $\emptyset \in \{\emptyset\} \in 2$, and $\emptyset \in 2$ as well, and its two elements are well-ordered: $\emptyset \in \{\emptyset\}$, or $0 \in 1$ if we prefer to write it this way.

Similarly, we can define $3 = \{0, 1, 2\}$, $4 = \{0, 1, 2, 3\}$ and so on – these are ordinals (and actually this is how the natural numbers are defined within set theory).

Basically, we get $n + 1$ as $n \cup \{n\}$. This works in general:

**Definition: $\alpha + 1$.**

If $\alpha$ is an ordinal, let $\alpha + 1$ denote the set $\alpha \cup \{\alpha\}$.

**Proposition**

If $\alpha$ is an ordinal, then so is $\alpha + 1$.

**Proof**

First we show that $\alpha + 1$ is transitive. Let $\gamma \in \beta \in \alpha + 1$. Then, either $\beta \in \alpha$ or $\beta = \alpha$. In the former case, since $\alpha$ is transitive, from $\gamma \in \beta \in \alpha$ we get $\gamma \in \alpha$, thus $\gamma \in \alpha \cup \{\alpha\} = \alpha + 1$ as well. In the latter case, when $\beta = \alpha$, we have $\gamma \in \beta = \alpha$, thus $\gamma \in \alpha \cup \{\alpha\} = \alpha + 1$ as well.

Now we show that $\alpha + 1$ is well-ordered by $\in$.

For irreflexivity, let $\beta \in \alpha + 1$. If $\beta \in \alpha$, then $\beta \notin \beta$ since $\alpha$ is well-ordered by $\in$. If $\beta = \alpha$, then we know that $\beta \notin \beta$, since $\alpha$ is an ordinal. Thus, $\in$ is irreflexive over $\alpha + 1$.

For transitivity, let $\beta_1, \beta_2, \beta_3 \in \alpha + 1$ be so that $\beta_1 \in \beta_2$ and $\beta_2 \in \beta_3$. If neither of them is $\alpha$, then they are in $\alpha$ as well which is well-ordered by $\in$. If $\beta_3 = \alpha$, then we have $\beta_1 \in \beta_2 \in \alpha$ which implies $\beta_1 \in \alpha$ since $\alpha$ is transitive. If $\beta_2 = \alpha$ and thus $\beta_3 \neq \alpha$ (otherwise it would be the case $\alpha \in \alpha$ ) then we have $\beta_3 \in \alpha$, thus $\beta_3 \in \alpha$ and $\alpha \in \beta_3$ both hold which is a contradiction, since by the transitivity of $\alpha$ we get then $\alpha \in \alpha$. Similarly, $\beta_1 = \alpha$ also gives
us a contradiction, since then $\beta_2 \in \alpha$ and $\alpha \in \beta_2$ both hold, thus $\alpha \in \alpha$ by transitivity of $\alpha$.

For trichotomy, let $\beta, \gamma \in \alpha + 1$. If both of them are in $\alpha$, then either $\beta \in \gamma$, $\gamma \in \beta$ or $\beta = \gamma$ since $\alpha$ is well-ordered. Otherwise, if $\beta = \gamma = \alpha$ then they are equal; if $\beta = \alpha$ and $\gamma \neq \alpha$, then $\gamma \in \alpha$; and if $\gamma = \alpha$ and $\beta \neq \alpha$, then $\beta \in \alpha$.

Finally, $\alpha + 1$ is well-ordered by $\in$. Assume there is an infinite descending chain $\ldots \in \beta_2 \in \beta_1 \in \beta_0$ with each $\beta_i$ being a member of $\alpha + 1$. Then if $\beta_i = \alpha$, then $i = 0$, since otherwise $\alpha \in \beta_{i-1}$ which is in turn a member of $\alpha + 1$, thus it’s either $\alpha$ or a member of $\alpha$ – both cases imply $\alpha \in \alpha$ which cannot happen. Thus, $\ldots \in \beta_2 \in \beta_1$ is an infinite descending chain in $\alpha$ which is a contradiction since $\alpha$ is well-ordered by $\in$.

In the previous examples all the elements of ordinals were ordinals themselves. This is not a coincidence:

**Proposition**

Elements of ordinals are ordinals.

**Proof**

Let $\alpha$ be an ordinal and $\beta \in \alpha$. By transitivity, all the elements of $\beta$ are elements of $\alpha$ as well, thus they are also well-ordered by $\in$. (It is clear that any sub-ordering of a well-order is a well-order.)

We still have to show that $\beta$ is transitive, so let $\delta \in \gamma \in \beta$. By $\alpha$ being transitive we get from $\gamma \in \beta \in \alpha$ that $\gamma \in \alpha$, which in turn implies by $\delta \in \gamma \in \alpha$ and again the transitivity of $\alpha$ that $\delta \in \alpha$ as well. Thus, since $\alpha$ is well-ordered by $\in$, and $\beta, \delta$ are two elements of $\alpha$, either $\beta = \delta$, or $\beta \in \delta$, or $\delta \in \beta$ has to hold. Now by $\delta \in \gamma \in \beta$, each of them being a member of $\alpha$ which is well-ordered by $\in$, $\beta = \delta$ cannot happen (that would violate irreflexivity). Also, $\beta \in \delta$ cannot happen (applying transitivity we would get $\beta \in \beta$, violating irreflexivity again). Thus it has to be the case $\delta \in \beta$, so $\beta$ is indeed a transitive set.

Also, in the above constructions for the finite ordinals we had that the “initial part” of an ordinal (say, $\{0, 1\}$ within $4 = \{0, 1, 2, 3\}$) was also an ordinal (in this case, 2). This is again not a coincidence:

**Definition: Initial segment of an ordinal.**

A subset $X$ of an ordinal $\alpha$ is called an initial segment of $\alpha$ if whenever $\beta \in X$ and $\gamma \in \beta$, then also $\gamma \in X$.

That is, an initial segment is “closed downwards” with respect to $\in$. Or, equivalently, it is a transitive subset of an ordinal.

**Proposition**

If $X$ is an initial segment of an ordinal, then either $X \in \alpha$ (and thus $X$ is an ordinal), or $X = \alpha$. 
First we show that \( X \) is an ordinal. \( X \) is transitive by definition. Also, since \( X \) is a subset of \( \alpha \) and \( \alpha \) is well-ordered by \( \in \), we have that \( X \) is also well-ordered by \( \in \).

Assume \( X \not= \alpha \). Then there is some element \( \beta \in \alpha - X \). Since \( \in \) is a well-ordering on \( \alpha \), we have either \( x \in \beta \) or \( \beta \not\in x \) or \( \beta = x \) for each \( x \in X \). But since \( X \) is transitive, \( \beta \in X \) would imply \( \beta \in X \), a contradiction; thus, since \( \beta = x \) is also not an option (as \( x \in X \) but \( \beta \not\in X \)), we have \( x \in \beta \) for all \( x \in X \). That is, if \( X \) is a transitive subset of \( \alpha \), then either \( X = \alpha \) or \( X \not\subseteq \beta \) for some \( \beta \in \alpha \), that is, a transitive subset of some ordinal \( \beta \in \alpha \).

Let us set \( \beta_0 = \beta \) and for each integer \( n \geq 0 \), let \( \beta_{n+1} \in \beta_n \) be an ordinal with \( X \subseteq \beta_{n+1} \) if such an ordinal exists. Note that since \( \beta_0 \in \alpha \) and \( \beta_{n+1} \in \beta_n \in \alpha \) implies \( \beta_{n+1} \in \alpha \) by transitivity of \( \alpha \), we have each \( \beta_n \) is a member of \( \alpha \). Now since \( \alpha \) is well-ordered, there is no infinite descending chain of such \( \beta \)'s, thus at some point we get \( X = \beta_n \), thus \( X \) is indeed an element of \( \alpha \).

The above claim is good since it helps proving that ordinals are well-ordered:

**Proposition**

Any set of ordinals is well-ordered by \( \in \).

**Proof**

We know that \( \alpha \not\in \alpha \) for any ordinal \( \alpha \), so irreflexivity is fine. Also, if \( \gamma \in \beta \in \alpha \) for the ordinals \( \alpha, \beta, \gamma \), then by the transitivity of \( \alpha \) we get \( \gamma \in \alpha \), proving that \( \in \) is transitive among ordinals.

For trichotomy, we use the previous proposition on initial segments. Let \( \alpha \) and \( \beta \) be ordinals, \( \alpha \not= \beta \).

Then \( \gamma = \alpha \cap \beta \) is transitive, since if \( \varepsilon \in \delta \in \gamma \), then by \( \gamma \subseteq \alpha \) we get \( \varepsilon \in \delta \in \alpha \), thus \( \varepsilon \in \alpha \) by transitivity of \( \alpha \); and similarly by \( \varepsilon \in \delta \in \beta \) we also have \( \varepsilon \in \beta \), thus \( \varepsilon \in \alpha \cap \beta = \gamma \), proving transitivity of \( \gamma \).

Hence, by the above proposition we get that either \( \gamma \in \alpha \) or \( \gamma = \alpha \), and either \( \gamma \in \beta \) or \( \gamma = \beta \). Now assume \( \gamma \in \alpha \) and \( \gamma \in \beta \) both hold. Then \( \gamma \in \alpha \cap \beta = \gamma \), which cannot happen since \( \gamma \) is an ordinal. Thus the following cases can hold:

- \( \gamma \in \alpha \) and \( \gamma = \beta \). In this case \( \beta \in \alpha \).
- \( \gamma = \alpha \) and \( \gamma \in \beta \). In this case \( \alpha \in \beta \).
- \( \gamma = \alpha \) and \( \gamma = \beta \). In this case \( \alpha = \beta \).

So \( \in \) is trichotome over ordinals.

Finally, assume there is an infinite descending chain \( \ldots \in \alpha_2 \in \alpha_1 \in \alpha_0 \) of ordinals. Then by the transitivity of \( \alpha_0 \) we get that every \( \alpha_n \) with \( n \geq 1 \) is a member of \( \alpha_0 \): \( \alpha_1 \in \alpha_0 \) by definition, and applying induction we have that if \( \alpha_n \in \alpha_0 \), then by \( \alpha_{n+1} \in \alpha_n \) and the transitivity of \( \alpha_0 \) we get \( \alpha_{n+1} \in \alpha_0 \). But then \( \alpha_0 \) is not well-ordered by \( \in \), a contradiction.

From now on we will use the notation \( \alpha < \beta \) instead of \( \alpha \in \beta \) when it is the ordering among the ordinals that matters.
At this point it might be unclear whether there are ordinals at all that are not those of the form \( n \) for \( n \geq 0 \), but there are.

**Proposition**

Assume \( X = \{ \alpha_i : i \in I \} \) is a set of ordinals. Then the union \( \alpha = \bigcup_{i \in I} \alpha_i \) is an ordinal, and is the supremum of this set (with respect to the well-ordering < defined between ordinals).

**Proof**

The set \( X \) either has a largest element or not. If it has a largest element \( \beta \), then for each \( \gamma \in X, \gamma \neq \beta \) we have \( \gamma < \beta \), that is, \( \gamma \in \beta \) implying \( \gamma \subseteq \beta \) since \( \beta \) is transitive. Hence in that case the union is \( \beta \), clearly still an ordinal, and it is indeed the supremum of this set, being its largest element.

Now assume \( X \) does not have a largest element.

Then for each \( i \in I \) we have \( \alpha_i \in \alpha \) since \( \alpha_i < \alpha_j \) for some \( j \in I \), and \( \alpha \) is the union of all the \( \alpha_j \), hence \( \alpha_i \in \alpha \) as well. Thus, if \( \alpha \) is an ordinal, then it is an upper bound of \( X \).

To see that \( \alpha \) is an ordinal, let \( \gamma \in \beta \in \alpha \). Since \( \gamma \) is the union of the sets \( \alpha_i \), \( \beta \in \alpha_i \) holds for some \( i \in I \), hence by transitivity of \( \alpha \) we get \( \gamma \in \alpha_i \), thus \( \gamma \in \alpha \) as well. Thus, \( \alpha \) is a transitive set.

To see that \( \alpha \) is well-ordered by \( \in \), we check all the properties. Assume \( x, y, z \in \alpha \). Then \( x \in \alpha_i, y \in \alpha_j \) and \( z \in \alpha_k \) for some \( i, j, k \in I \). Since any set of ordinals is well-ordered by \( \in \), there is a greatest element in the set \( \{ \alpha_i, \alpha_j, \alpha_k \} \), let it be \( \alpha_i \). Then by transitivity we get \( x, y, z \in \alpha_i \). Thus, since \( \alpha_i \) is well-ordered by \( \in \), we get that \( x \notin x, x \in y \) and \( y \in z \) imply \( x \in z \) and exactly one of \( x \in y, y \in x \) or \( x = y \) holds. Thus \( \alpha \) is strictly linearly ordered by \( \in \). Now assume there is an infinite descending chain \( \ldots < x_2 < x_1 < x_0 \) of elements of \( \alpha \). Then since \( \alpha \) is the union of the \( \alpha_i \) sets, \( x_0 \in \alpha_i \) for some \( i \in I \). By transitivity of \( \alpha_i \) we get that \( x_{n+1} \in x_n \in \alpha_i \) implies \( x_{n+1} \in \alpha_i \), thus in that case \( \alpha_i \) also contains an infinite descending chain, which is a contradiction since \( \alpha_i \) is also well-ordered by \( \in \).

Hence, \( \alpha \) is an upper bound of the ordinals \( \alpha_i \). Now assume \( \beta \) is also an upper bound. Then by \( \alpha_i < \beta \) for each \( i \in I \), that is, \( \alpha_i \in \beta \) and \( \beta \) being transitive implies \( \alpha_i \subseteq \beta \) for each \( i \in I \). Thus, their union \( \alpha \) is also a subset of \( \beta \). Moreover, \( \alpha \) is transitive, thus either \( \alpha = \beta \), or \( \alpha \in \beta \), that is, \( \alpha < \beta \), thus \( \alpha \) is indeed the supremum of the set \( X \) of ordinals.

Hence, \( \omega = \{ 0, 1, 2, \ldots \} \) is an ordinal. (This is the same \( \omega \) as the one we used previously for indexing.)

We have seen two constructions for constructing larger ordinals from smaller ones: the construction \( \alpha \mapsto \alpha + 1 \) and taking union of a set of ordinals. The following proposition states that these are essentially the only methods for constructing ordinals.

**Proposition**

Every ordinal \( \alpha \) is either a successor ordinal, that is, an ordinal of the form \( \beta + 1 \), or a limit ordinal, that is, an ordinal of the form \( \bigvee_{\beta<\alpha} \beta \).
We know that $\alpha$ is well-ordered by $\in$, which is the ordering relation among ordinals and elements of $\alpha$ are ordinals as well.

Now either $\alpha$ has a largest element with respect to $\in$, or it does not have. If $\beta$ is the largest element of $\alpha$, then $\alpha = \beta + 1$. Indeed, since $\beta \in \alpha$ and $\alpha$ is transitive, $\alpha$ contains all the members of $\beta$, that is, $\beta \subseteq \alpha$. Moreover, if $\gamma \in \alpha - \beta$, then either $\gamma \in \beta$, $\beta \in \gamma$ or $\beta = \gamma$; now since $\gamma \in \alpha - \beta$, the case $\gamma \in \beta$ is ruled out; since $\beta$ is the largest element of $\alpha$, the case $\beta \in \gamma$ is also ruled out, thus if $\gamma \in \alpha - \beta$, then $\gamma = \beta$, which means precisely that $\alpha = \beta + 1$.

Now assume $\alpha$ does not have a largest element with respect to $\in$. Then by transitivity, each member $\beta \in \alpha$ is a subset of $\alpha$, that is, $\beta < \alpha$ implies $\beta \subseteq \alpha$, thus $\bigcup_{\beta < \alpha} \beta \subseteq \alpha$.

Also, for each element $\gamma \in \alpha$ there is an ordinal $\beta \in \alpha$ with $\gamma < \beta$ (since no $\gamma$ is a largest element of $\alpha$), hence each element $\gamma$ of $\alpha$ is a member of some $\beta < \alpha$, hence $\alpha$ itself is a subset of their union: $\alpha \subseteq \bigcup_{\beta < \alpha} \beta$, and these two statements together imply $\alpha = \bigcup_{\beta < \alpha} \beta$.

It is also clear that $\alpha$ is the supremum of the set $\{\beta : \beta < \alpha\}$ in this case. (Note that $\beta < \alpha$ is $\beta \in \alpha$ here, thus the latter set is $\alpha$ itself.)

Technically, $0 = \bigvee \emptyset$ is often not viewed as a limit ordinal but as a separate case, that is, there are three types of ordinals: zero, successor ordinals and limit (nonzero) ordinals, and these three cases are mutually exclusive.

Also, $\alpha + 1$ is denoted this way by a reason:

**Proposition**

Let $\alpha < \beta$ be ordinals. Then $\alpha + 1 \leq \beta$.

**Proof**

Since $<$ is trichotome over the ordinals, we have either $\alpha + 1 = \beta$, $\alpha + 1 < \beta$ or $\beta < \alpha + 1$.

But if $\beta < \alpha + 1$, that is, $\beta \in \alpha \cup \{\alpha\}$, then either $\beta \in \alpha$ (i.e. $\beta < \alpha$) or $\beta = \alpha$, both contradicting to $\alpha < \beta$. Hence, $\alpha + 1 \leq \beta$.

Summarizing some properties of ordinals we get:

- 0 is an ordinal.
- There is a well-ordering relation $<$ over the ordinals.
- For each ordinal $\alpha$, there is an ordinal denoted $\alpha + 1$.
- It holds that $\alpha < \alpha + 1$ and for each $\alpha < \beta$ we have $\alpha + 1 \leq \beta$.
- For any ordinal $\alpha$, the collection of ordinals smaller than $\alpha$ is a set$^6$.
- For every ordinal $\alpha$, one of the following three mutually exclusive cases hold:
  - $\alpha = 0$.

$^6$Actually, this set is $\alpha$ itself.
- $\alpha = \beta + 1$ for some ordinal $\beta$.
- $\alpha$ is a nonzero limit ordinal, that is, $\alpha = \bigvee_{\beta<\alpha} \beta$.

These will be enough to show that a monotone function over a complete lattice always has a least pre-fixed point.

**The Kleene Fixed Point Theorem**

As the collection of the ordinals is well-ordered, we can do well-founded induction over the ordinals, e.g., defining a sequence, indexed by ordinals, within some set $P$, where each element is defined based on the earlier elements of the sequence.

**Proposition: Kleene Fixed Point Theorem.**

Let $P$ be a complete lattice and $f : P \to P$ be a monotone function. We define to each ordinal $\alpha$ the following element $x_\alpha$ of $P$: $x_0 = \bot$; if $\alpha = \beta + 1$ is a successor ordinal, then $x_\alpha = f(x_\beta)$ and if $\alpha = \bigvee_{\beta<\alpha} \beta$ is a nonzero limit ordinal, then let $x_\alpha = \bigvee_{\beta<\alpha} x_\beta$.

Then, for some ordinal $\alpha$, $x_\alpha$ is the least (pre)fixed point of $f$.

In the proof we will use well-founded induction.

**Proof**

Let $X \subseteq P$ be the set of those elements occurring in the sequence above (i.e. $x \in X$ if and only if $x = x_\alpha$ for some ordinal $\alpha$). Then $X$ contains $\bot$, since $x_0 = \bot$, and if $x \in X$, then $f(x) \in X$ as well (since if $x = x_\alpha$, then $f(x) = x_{\alpha+1}$ by definition).

Also, each $x_\alpha$ is a post-fixed point of $f$, which can be shown via well-founded induction. For $\alpha = 0$ we have $x_\alpha = \bot$ which is a post-fixed point. For successor ordinals $\alpha = \beta + 1$, if $x_\beta$ is a post-fixed point, then $x_\alpha = f(x_\beta)$ is also a post-fixed point since monotone functions transform post-fixed points into post-fixed points. For limit non-zero ordinals $\alpha = \bigvee_{\beta<\alpha} \beta$, we have that a supremum of post-fixed points is still a post-fixed point. To see that, let $Y \subseteq P$ be a set of post-fixed points. Then for each $y \in Y$ we have $y \leq \bigvee Y$, thus by monotonicity we get $y \leq f(y) \leq f(\bigvee Y)$, that is, $f(\bigvee Y)$ is an upper bound for $Y$ while $\bigvee Y$ is the least upper bound, so $\bigvee Y \leq f(\bigvee Y)$, the supremum is also a post-fixed point.

We also claim that $\bigvee X \in X$, that is, $X$ has a greatest element. To see this, let us fix to each $x \in X$ an ordinal $\alpha(x)$ with $x = x_{\alpha(x)}$ and let $\alpha$ be the supremum of these ordinals (as it’s the supremum of a set of ordinals, $\alpha$ is also an ordinal). To circumvent a case analysis, let $\gamma$ be a limit ordinal larger than $\alpha$ (such ordinal exists, e.g. the supremum of the ordinals $\alpha$, $\alpha + 1$, $\alpha + 2$, ...) Then $x_\gamma = \bigvee_{\beta<\gamma} x_\beta$ and since each $x \in X$ appears as $x = x_{\alpha(x)}$ for some $\alpha(x) < \gamma$, we get that $\{x_\beta : \beta < \gamma\}$ is $X$. Hence, $x_\gamma = \bigvee X$ and since $\gamma$ is an ordinal, $\bigvee X \in X$, that is, $X$ has a largest element $x = x_\gamma$.

But then $x_\gamma$ is still a post-fixed point of $f$, thus $x_\gamma \leq f(x_\gamma) = x_{\gamma+1}$ which is also in $X$ since $\gamma + 1$ is also an ordinal. But since $x_\gamma$ is the largest element of $X$, we have that $x_{\gamma+1} \leq x_\gamma$ as well, thus $x_\gamma = x_{\gamma+1} = f(x_\gamma)$, hence $x_\gamma$ is a fixed point of $f$.

It remains to show that $x_\gamma$ is the least pre-fixed point. We show by well-founded induction...
that whenever $x$ is a pre-fixed point of $f$, and $\alpha$ is an ordinal, then $x_\alpha \leq x$.

For $\alpha = 0$ the claim holds, since $x_0 = \perp \leq x$ for any $x \in P$.

If $\alpha = \beta + 1$ is a successor ordinal, then by induction we have $x_\beta \leq x$. By monotonicity of $f$ we get $x_\alpha = x_{\beta+1} = f(x_\beta) \leq f(x)$ but since $x$ is a pre-fixed point, we also have $f(x) \leq x$, implying $x_\alpha \leq x$.

Finally, if $\alpha = \bigvee_{\beta < \alpha} \beta$ is a limit ordinal, then by the induction hypothesis each $x_\beta$ with $\beta < \alpha$ is a lower bound of $x$, that is, $x$ is an upper bound of the set $\{x_\beta : \beta < \alpha\}$. Since $x_\alpha = \bigvee_{\beta < \alpha} x_\beta$ is the least upper bound, we get $x_\alpha \leq x$ also in this case.

Hence, $x_\gamma$ is the least (pre)fixed point of $f$ for some ordinal $\gamma$. (And for each ordinal $\delta > \gamma$, $x_\delta = x_\gamma$ from that point.)