3.2 Predicate Logic

While propositional logic is a good starting point, most interesting statements in mathematics contain variables over domains larger than simply $\{\text{True}, \text{False}\}$. For example, the statement "x is a power of 2" is not a proposition because its truth value depends on the value of x. It is only after we *substitute* a value for x that we may determine whether the resulting statement is True or False. For example, if $x = 8$, then the statement becomes "8 is a power of 2", which is True. But if $x = 7$, then the statement becomes "7 is a power of 2", which is False.

A statement whose truth value depends on one or more variables from any set is a **predicate**. Formally, a predicate is a *function* whose codomain is $\{\text{True}, \text{False}\}$ (and whose domain could be any set). We typically use uppercase letters starting from P to represent predicates, differentiating them from propositional variables. For example, if $P(x)$ is defined to be the statement "x is a power of 2", then $P(8)$ is True and $P(7)$ is False. Thus a predicate is like a proposition except that it contains one or more variables; when we substitute particular values for the variables, we obtain a proposition.

As with all functions, predicates can depend on more than one variable. For example, if we define the predicate $Q(x, y)$ to mean "$x^2 = y$," then $Q(5, 25)$ is True since $5^2 = 25$, but $Q(5, 24)$ is False.[1]

[1] Just as how common arithmetic operators like $+$ are really binary functions, the common comparison operators like $=$ and $<$ are *binary predicates*, taking two numbers and returning True or False.

We usually define a predicate by giving the statement that involves the variables, e.g., "$P(x)$ is the statement 'x is a power of 2.'" However, there is another component which is crucial to the definition of a predicate: the domain that each of the predicate's variable(s) belong to. You must always give the domain of a predicate as part of its definition. So we would complete the definition of $P(x)$ as follows:

$$P(x) : \text{`}x \text{ is a power of 2,' where } x \in \mathbb{N}.$$

Quantification of variables

Unlike propositional formulas, a predicate by itself does not have a truth value: as we discussed earlier, "x is a power of 2" is neither True nor False, since we don't know the value of x. We have seen one way to obtain a truth value in substituting a concrete element of the predicate's domain for its input, e.g., setting $x = 8$ in the statement "x is a power of 2," which is now True.

However, we often don't care about whether a specific value satisfies a predicate, but rather some aggregation of the predicate's truth values over *all* elements of its domain. For example, the statement "every real number x satisfies the inequality $x^2 - 2x + 1 \geq 0$" doesn't make a claim about a specific real number like 5 or π, but rather *all possible* values of x!

There are two types of "truth value aggregation" we want to express; each type is represented by a **quantifier** that modifies a predicate by specifying how a certain variable should be interpreted.

Existential quantifier

Definition. The **existential quantifier** is written as $\exists$, and represents the concept of "*there exists* an element in the domain that satisfies the given predicate."

Example. For example, the statement $\exists x \in \mathbb{N},\ x \geq 0$ can be translated as "there exists a natural number x that is greater than or equal to zero." This statement is True since (for example) when $x = 1$, we know that $x \geq 0$.

Note that there are many more natural numbers that are greater than or equal to 0. The existential quantifier says only that there has to be *at least one* element of the domain satisfying the predicate, but it doesn't say exactly how many elements do so.

One should think of $\exists x \in S$ as an abbreviation for a big **OR** that runs through all possible values for x from the domain S. For the previous example, we can expand it by substituting all possible natural numbers for x:[2]

[2] In this case, the **OR** expression is technically infinite, since there are infinitely many natural numbers.

$$(0 \geq 0) \lor (1 \geq 0) \lor (2 \geq 0) \lor (3 \geq 0) \lor \cdots$$

Universal quantifier

Definition. The **universal quantifier** is written as $\forall$, and represents the concept that "*every* element in the domain satisfies the given predicate."

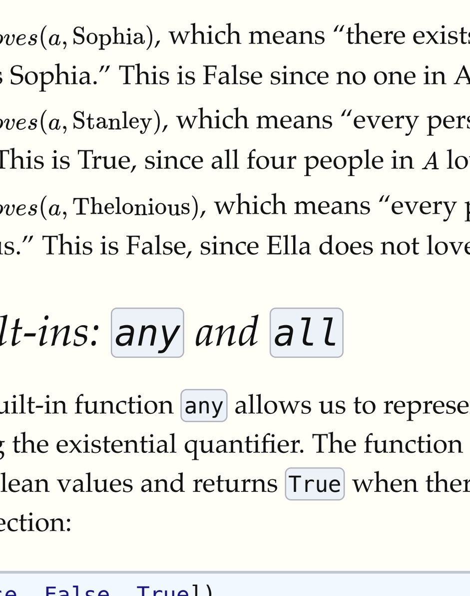
Example. For example, the statement $\forall x \in \mathbb{N},\ x \geq 0$ can be translated as "every natural number x is greater than or equal to zero." This statement is True since the smallest natural number is zero itself. However, the statement $\forall x \in \mathbb{N},\ x \geq 10$ is False, since not every natural number is greater than or equal to 10.

One should think of $\forall x \in S$ as an abbreviation for a big **AND** that runs through all possible values of x from S. Thus, $\forall x \in \mathbb{N},\ x \geq 0$ is the same as

$$(0 \geq 0) \land (1 \geq 0) \land (2 \geq 0) \land (3 \geq 0) \land \cdots$$

Example. Let us look at a simple example of these quantifiers. Suppose we define $Loves(a, b)$ to be a binary predicate that is True whenever person a loves person b.

For example, the diagram below defines the relation "Loves" for two collections of people: A = {Ella, Patrick, Malena, Breanna}, and B = {Laura, Stanley, Thelonious, Sophia}. A line between two people indicates that the person on the left loves the person on the right.



Consider the following statements.

- $\exists a \in A,\ Loves(a, \text{Thelonious})$, which means "there exists someone in A who loves Thelonious." This is True since Malena loves Thelonious.[3]
- $\exists a \in A,\ Loves(a, \text{Sophia})$, which means "there exists someone in A who loves Sophia." This is False since no one in A loves Sophia.
- $\forall a \in A,\ Loves(a, \text{Stanley})$, which means "every person in A loves Stanley." This is True, since all four people in A love Stanley.
- $\forall a \in A,\ Loves(a, \text{Thelonious})$, which means "every person in A loves Thelonious." This is False, since Ella does not love Thelonius.

[3] We could also have said here that Breanna loves Thelonious.

Python built-ins: `any` and `all`

In Python, the built-in function `any` allows us to represent logical statements using the existential quantifier. The function `any` takes a collection of boolean values and returns `True` when there exists a `True` value in the collection:

```
>>> any([False, False, True])
True
>>> any([])  # An empty collection has no True values!
False
```

This might not seem useful by itself, but remember that we can use comprehensions to transform one collection of data into another. For example, suppose we are given a set of strings S and wish to determine whether any of them start with the letter `'D'`. In predicate logic, we could write this as the statement $\exists s \in S,\ s[0] = \text{`D'}$. And in Python, we could do the following:

```
>>> strings = ['Hello', 'Goodbye', 'David']
>>> any([s[0] == 'D' for s in strings])
True
```

This example serves to highlight several elegant parallels between our mathematical statement and equivalent Python expression:

- $\exists$ corresponds to calling the `any` function
- $s \in S$ corresponds to `for s in strings`[4]
- $s[0] = \text{`D'}$ corresponds to `s[0] == 'D'`

[4] The naming conventions are a bit different, however: in mathematics, we tend to represent collections using capital letters, whereas in Python all variables are lower-case words.

Python includes another built-in function `all` that can be used as a universal quantifier. The `all` function is given a collection of values and evaluates to `True` when every element has the value `True`. For example, if we wanted to express $\forall s \in S,\ s[0] = \text{`D'}$ in Python, we could write:

```
>>> strings = ['Hello', 'Goodbye', 'David']
>>> all([s[0] == 'D' for s in strings])
False
```

Of course, Python is more limited than mathematics because there are limits on the size of the collections, and so we cannot easily express existential statement quantified over infinite domains like $\mathbb{N}$ or $\mathbb{R}$. We'll discuss this in more detail in a later section.

Writing sentences in predicate logic

Now that we have introduced the existential and universal quantifiers, we have a complete set of tools needed to represent all statements we'll see in this course. A general formula in predicate logic is built up using the existential and universal quantifiers, the propositional operators $\neg$, $\land$, $\lor$, $\Rightarrow$, and $\Leftrightarrow$, and arbitrary predicates. To ensure that the formula has a fixed truth value, we will require every variable in the formula to be quantified.[5] We call a formula with no unquantified variables a **sentence**. So for example, the formula

[5] Other texts will often refer to quantified variables as *bound variables*, and unquantified variables as *free variables*.

$$\forall x \in \mathbb{N},\ x^2 > y$$

is not a sentence: even though x is quantified, y is not, and so we cannot determine the truth value of this formula. If we quantify y as well, we get a sentence:

$$\forall x, y \in \mathbb{N},\ x^2 > y.$$

However, don't confuse a formula being a sentence with a formula being True! As we'll see repeatedly throughout the course, it is quite possible to express both True and False sentences, and part of our job will be to determine whether a given sentence is True or False, and to prove it.

Commas: avoid them!

Here is a common question from students who are first learning symbolic logic: "does the comma mean 'and' or 'then'?" As we discussed at the start of the course, we study to predicate logic to provide us with an unambiguous way of representing ideas. The English language is filled with ambiguities that can make it hard to express even relatively simple ideas, much less the complex definitions and concepts used in many fields of computer science. We have seen one example of this ambiguity in the English word "or," which can be inclusive or exlusive, and often requires additional words of clarification to make precise. In everyday communication, these ambiguous aspects of the English language contribute to its richness of expression. But in a technical context, ambiguity is undesirable: it is much more useful to limit the possible meanings to make them unambiguous and precise.

There is another, more insidious example of ambiguity with which you are probably more familiar: the *comma*, a tiny, easily-glazed-over symbol that people often infuse with different meanings. Consider the following statements:

1. If it rains tomorrow, I'll be sad.
2. David is cool, Toniann is cool.

Our intuitions tell us very different things about what the commas mean in each case. In the first, the comma means *then*, separating the hypothesis and conclusion of an implication. But in the second, the comma is used to mean *and*, the implicit joining of two separate sentences.[6] The fact that we are all fluent in English means that our prior intuition hides the ambiguity in this symbol, but it is quite obvious when we put this into the more unfamiliar context of predicate logic, as in the formula:

[6] Grammar-savvy folks will recognize this as a *comma splice*, which is often frowned upon but informs our reading nonetheless.

$$P(x), Q(x)$$

This, of course, is where the confusion lies, and is the origin of the question posed at the beginning of this section. Because of this ambiguity, **never use the comma to connect propositions**. We already have a rich enough set of symbols, including $\land$ and $\Rightarrow$, and do not need another one that is both ambiguous and adds nothing new!

That said, keep in mind that commas do have two valid uses in predicate formulas:

- immediately after a variable quantification, or separating two variables with the same quantification
- separating arguments to a predicate function

You can see both of these usages illustrated below, but please do remember that these are the *only* valid places for the comma within symbolic notation!

$$\forall x, y \in \mathbb{N},\ \forall z \in \mathbb{R},\ P(x, y) \Rightarrow Q(x, y, z)$$

Manipulating negation

We have already seen some equivalences among logical formulas, such as the equivalence of $p \Rightarrow q$ and $\neg p \lor q$. While there are many such equivalences, the only other major type that is important for this course are the ones used to simplify negated formulas. Taking the negation of a statement is extremely common, because often when we are trying to decide if a statement is True, it is useful to know exactly what its negation means and decide whether the negation is more plausible than the original.

Given any formula, we can state its negation simply by preceding it by a $\neg$ symbol:

$$\neg\big(\forall x \in \mathbb{N},\ \exists y \in \mathbb{N},\ x \geq 5 \lor x^2 - y \geq 30\big).$$

However, such a statement is rather hard to understand if you try to transliterate each part separately: "Not for every natural number x, there exists a natural number y, such that x is greater than or equal to 5 or $x^2 - y$ is greater than or equal to 30."

Instead, given a formula using negations, we apply some *simplification rules* to "push" the negation symbol to the right, closer the to individual predicates. Each simplification rule shows how to "move the negation inside" by one step, giving a pair of equivalent formulas, one with the negation applied to one of the logical operator or quantifiers, and one where the negation is applied to inner subexpressions.

- $\neg(\neg p)$ becomes p.
- $\neg(p \lor q)$ becomes $(\neg p) \land (\neg q)$.[7]
- $\neg(p \land q)$ becomes $(\neg p) \lor (\neg q)$.
- $\neg(p \Rightarrow q)$ becomes $p \land (\neg q)$.[8]
- $\neg(p \Leftrightarrow q)$ becomes $(p \land (\neg q)) \lor ((\neg p) \land q))$.
- $\neg(\exists x \in S,\ P(x))$ becomes $\forall x \in S,\ \neg P(x)$.
- $\neg(\forall x \in S,\ P(x))$ becomes $\exists x \in S,\ \neg P(x)$.

[7] The negation rules for AND and OR are known as *deMorgan's laws*.

[8] Since $p \Rightarrow q$ is equivalent to $\neg p \lor q$.

It is usually easy to remember the simplification rules for $\land$, $\lor$, $\forall$, and $\exists$, since you simply "flip" them when moving the negation inside. The intuition for the negation of $p \Rightarrow q$ is that there is only one case where this is False: when p has occurred but q does not. The intuition for the negation of $p \Leftrightarrow q$ is to remember that $\Leftrightarrow$ can be replaced with "have the same truth value," so the negation is "have different truth values."

What about the quantifiers? Consider a statement of the form $\neg(\exists x \in S,\ P(x))$, which says "there does not exist an element x of S that satisfies P." The only way this could be true is for every element of S to *not* satisfy P: "every element x of S does not satisfy P." A similar line of reasoning applies to $\neg(\forall x \in S,\ P(x))$.