CHAPTER 3

First(-order) things first

With the ghost of George Boole finally satisfied, you're now ready to try your luck at sleeping once again. As you close your eyes, a wild amount of noise starts coming in through the window of your room. You get up, look outside, and you see Tarski's tank going up and down the street. Confused and choosing to ignore the (logical) consequences, you decide to think a bit more about what you just learned. All the limitations of the Boolean world become suddenly apparent...

1. Syntax still means how we write things down

1.1. First steps. Let's start with our basic symbols. We already learned about some of them in the previous chapter. But now, as we saw previously, we will introduce more, namely, we will introduce **quantifiers**:

- $\exists$ (syntactically *"rotated E"*, semantically *"exists"*);
- $\forall$ (syntactically *"rotated A"*, semantically *"for all"*);

But we also need things to quantify *over*, things like the property of mortality or of humanity or whatever else. Since we're trying, at the end of the day, to also do some mathematics, we will also think about functions.

Definition 1.1.1. A *first-order language* is a set of symbols $\mathcal{L}$ which consists of two disjoint subsets:

(1) The *logical symbols* of $\mathcal{L}$:[1]

  (a) A countable set of *variables* Var, which we will usually denote by $x, x_1, x_2, \dots, y, y_1, y_2, \dots, z, z_1, z_2, \dots$, etc.

  (b) Our good old logical symbols: $\wedge, \vee, \neg, \rightarrow, \leftrightarrow, \forall, \exists$.

  (c) A symbol for equality $\doteq$.

[1]This part is appears unchanged in EVERY first-order language $\mathcal{L}$.

(2) The *signature* of $\mathcal{L}$ (aka the *non-logical symbols*) of $\mathcal{L}$, these are the following three $\underline{\text{disjoint}}$ sets:

(a) A set of *constant symbols* $\mathsf{Const}(\mathcal{L})$. We will usually denote (abstract) constant symbols as $\underline{c}, \underline{c}_1, \underline{c}_2, \ldots$.

(b) A set of *relation symbols* $\mathsf{Rel}(\mathcal{L})$. We will usually denote (abstract) relation symbols as $\underline{R}, \underline{R}_1, \underline{R}_2, \ldots$.

(c) A set of *function symbols* $\mathsf{Fun}(\mathcal{L})$. We will usually denote (abstract) function symbols as $\underline{f}, \underline{f}_1, \underline{f}_2, \ldots$.

As part of the signature, we also have an *arity* function

$$\mathsf{arity}_{\mathcal{L}} : \mathsf{Rel}(\mathcal{L}) \sqcup \mathsf{Fun}(\mathcal{L}) \to \mathbb{N}_{\geq 1}.$$

Let $\underline{R} \in \mathsf{Rel}(\mathcal{L})$. If $\mathsf{arity}(\underline{R}) = n$ then we say that $\underline{R}$ is an *n-ary relation symbol*. Similarly, if $\underline{f} \in \mathsf{Fun}(\mathcal{L})$. and $\mathsf{arity}(\underline{f}) = n$ then we say that $\underline{f}$ is an *n-ary function symbol*.

To make our lives a bit simpler and avoid transfinite things and Zorn, we will, unless otherwise stated, assume that $|\mathsf{Const}(\mathcal{L})|, |\mathsf{Rel}(\mathcal{L})|, |\mathsf{Fun}(\mathcal{L})| \leq \aleph_0$, i.e. that our **signatures are countable**. I'm underlying things here to make sure you understand that these are not constants relations or functions in the sense you may be familiar, these are constant relation or function $\underline{\text{symbols}}$.[2]

Again, just like when we were talking about the syntax of propositional logic, everything we will discuss in this section is purely syntactic; it's a bunch of strings of letters on a page (or a board) and will have NO MEANING yet.

Ah also, since for any first-order language $\mathcal{L}$, the logical symbols of $\mathcal{L}$ are always the same, we will be identifying a language with its signature.

Our first goal is to learn how to write formulas in a given first-order language $\mathcal{L}$. It'll take a minute, but we'll get there. Many things that we will be discussing in this section, have already shown up here and there when we were playing around with propositional logic (cf. the definition of $\mathcal{F}$-terms).

Definition 1.1.2. Let $\mathcal{L}$ be a first-order language. The *terms of* $\mathcal{L}$ (or *$\mathcal{L}$-terms*) are defined inductively, as follows:

(1) Every variable is an $\mathcal{L}$-term.

[2]Slowly as we understand the interplay between syntax and semantics (next section), I may every now and again start dropping the underlines, and probably so will you, but for now I'd like to keep them here.

(2) Every constant symbol $\underline{c} \in \mathsf{Const}(\mathcal{L})$ is an $\mathcal{L}$-term.

(3) If $\underline{f} \in \mathsf{Fun}(\mathcal{L})$ is an n-ary function symbol and $t_1, \dots, t_n$ are $\mathcal{L}$-terms, then $\underline{f}(t_1, \dots, t_n)$ is an $\mathcal{L}$-term.

(4) That's it – all terms are constructed by finitely many applications of (1), (2) and (3).

Remark 1.1.3. Relation symbols don't show up in term building!

Lemma 1.1.4. *Let $\mathcal{L}$ be a first-order language. Let $\mathcal{T}_0$ be the set $\mathsf{Var}(\mathcal{L}) \cup \mathsf{Const}(\mathcal{L})$. Inductively, define $\mathcal{T}_{n+1}$ to be the set:*

$$\mathcal{T}_{n+1} := \mathcal{T}_n \cup \{\underline{f}(t_1, \dots, t_k) : k \in \mathbb{N}, \underline{f} \in \mathsf{Fun}(\mathcal{L}), \mathsf{arity}(\underline{f}) = k, t_i \in \mathcal{T}_n\},$$

and let $\mathcal{T} = \bigcup_{n \in \mathbb{N}} \mathcal{T}_n$. Then $\mathcal{T}$ is the set of all $\mathcal{L}$-terms.

Proof. It should hopefully be clear that the set of all $\mathcal{L}$-terms is contained in $\mathcal{T}$. For the other inclusion, we show, by induction on $n \in \mathbb{N}$ that $\mathcal{T}_n$ is contained in the set of $\mathcal{L}$-terms. Indeed this is clear for $\mathcal{T}_0$ and follows by (3) for $n \geq 1$. $\square$

By the lemma above, the set of all $\mathcal{L}$-terms is $\mathcal{T} = \bigcup_{n \in \mathbb{N}} \mathcal{T}_n$. Observe that if $t \in \mathcal{T}$, then there is some minimal $n \in \mathbb{N}$ such that $t \in \mathcal{T}_n$. We call this n the **height** of t (some people use the word **length**, I may do so too from time to time). More down to earth, the length of any variable or constant symbol is 0 and the length of any other term is precisely the longest chain of function symbols that appears in it. Here is an example:

Example 1.1.5. Let $\mathcal{L}$ be a first-order language with a single **binary** (i.e. of arity 2) function symbol $\underline{g}$ and a single constant symbol $\underline{c}$. To compute the length of a term, we can break it up into a tree, as follows:

$$\begin{tikzcd}
 & & & \underline{g}(\underline{g}(x_1, \underline{c}), \underline{g}(\underline{g}(x_2, x_3), \underline{c})) \arrow[dll] \arrow[drr] & & & \\
 & \underline{g}(x_1, \underline{c}) \arrow[dl] \arrow[dr] & & & & \underline{g}(\underline{g}(x_2, , x_3)\underline{c}) \arrow[dl] \arrow[dr] & \\
x_1 & & \underline{c} & & \underline{g}(x_2, x_3) \arrow[dl] \arrow[dr] & & \underline{c} \\
 & & & x_2 & & x_3 &
\end{tikzcd}$$

The term above has height 3.

This isn't all that complicated, really, but to make sure you get it, you can try out the following:

Exercise 1.1.6. Let $\mathcal{L}$ be a language with a single constant symbol $\underline{c}$, one **unary** (i.e. of arity 1) function symbol $\underline{f}$ and one binary function symbol $\underline{g}$. Write down all $\mathcal{L}$-terms of length at most 3 in the variables x and y.

Just like matter is built of atoms and stuff, formulas are built out of inseparable components, the atoms,[3] so to speak, of first-order syntax, which we define below:

Definition 1.1.7. The *atomic formulas of* $\mathcal{L}$ (or *atomic* $\mathcal{L}$*-formulas*) are defined inductively as follows:

(1) If t_1, t_2 are $\mathcal{L}$-terms, then $t_1 \doteq t_2$ is an atomic $\mathcal{L}$-formula.[4]

(2) If $\underline{R} \in \mathsf{Rel}(\mathcal{L})$ is an n-ary relation. symbol and $t_1, \dots, t_n$ are terms, then $\underline{R}(t_1, \dots, t_n)$ is an atomic $\mathcal{L}$-formula.

Observe that, whereas in the definition of terms we mixed smaller terms to build bigger ones, in the definition of atomic formulas, we don't mix atomic formulas to build bigger ones. Atomic formulas are atoms indeed! They will form our base case in the definition of formulas proper.

Indeed, in the definition of formulas, atomic formulas will (up to a point of analogy obscuring reality) play the role that propositional variables played in the definition of propositional formulas.

Definition 1.1.8. The *formulas* of $\mathcal{L}$ (or $\mathcal{L}$-formulas) are defined inductively, as follows:

(1) Every atomic $\mathcal{L}$-formula is an $\mathcal{L}$-formula.

(2) If ϕ and ψ are $\mathcal{L}$-formulas, and $x \in \mathsf{Var}$, then:

(a) $(\phi \wedge \psi)$	(c) $(\phi \to \psi)$	(e) $(\forall x)\phi$
(b) $(\phi \vee \psi)$	(d) $(\neg\psi)$	(f) $(\exists x)\phi$

are all $\mathcal{L}$-formulas

(3) That's it – all $\mathcal{L}$ formulas are constructed by finitely many applications of (1) and (2), above.

[3]From α- (a-, "not") + $\tau\epsilon\mu\nu\omega$ (témno, "I cut"). Bet you weren't expecting a lesson in etymology in these notes.

[4]Note here that we use $\doteq$ because these are syntactic objects! Just like with the underlining business, as we go along, I may start dropping the dot above the equality symbol.

In analogy with Lemma 1.1.4 we have the following:

Exercise 1.1.9 (Unique reading of formulas)**.** Let $\mathcal{L}$ be a first-order language. Let $\mathcal{F}_0$ be the set of atomic $\mathcal{L}$-formulas. Inductively, define $\mathcal{F}_{n+1}$ to be the set:

$$\mathcal{F}_{n+1} := \mathcal{F}_n \cup \{\phi \wedge \psi, \phi \vee \psi, \phi \to \psi, \neg\phi, (\forall x)\phi, (\exists x)\phi : \phi, \psi \in \mathcal{F}_n, x \in \mathsf{Var}\}$$

and let $\mathcal{F} = \bigcup_{n\in\mathbb{N}} \mathcal{F}_n$. Prove that $\mathcal{F}$ is the set of all $\mathcal{L}$-formulas.

This allows us to define the **height** of an $\mathcal{L}$-formula, as the minimal $n \in \mathbb{N}$ such that $\phi \in \mathcal{F}_n$. Formulas are built in trees, just like terms. Below are some examples that we could certainly care about:

Example 1.1.10.

(1) The **language of graphs**, $\mathcal{L}_{\mathsf{graph}}$, consists of a single binary relation symbol $\underline{E}$.

(2) The **language of groups**, $\mathcal{L}_{\mathsf{grp}}$, consists of a constant symbol $\underline{1}$, a binary function symbol $\underline{\times}$ (usually we write $x\underline{\times}y$ for $\underline{\times}(x,y)$, because we are not maniacs), and a unary function symbol $\underline{^{-1}}$ (again for similar reasons, we usually write $x\underline{^{-1}}$ for $\underline{^{-1}}(x)$).

(3) The language of **Peano arithmetic**, $\mathcal{L}_{Peano}$ (which will be our main focus in the last chapter) consists of a constant symbol $\underline{0}$, a unary function symbol $\underline{S}$, and two binary function symbols $\underline{+}$ and $\underline{\times}$.

Exercise 1.1.11.

(1) Write two different $\mathcal{L}_{\mathsf{graph}}$-formulas of height 4 in this language.

(2) Write down two $\mathcal{L}_{\mathsf{grp}}$-formulas that use all symbols in the language at least twice.

1.2. Syntax can be annoying, I.

In this section, we will handle some of the syntactical quirks of first-order logic. Thankfully, we have gotten some practice with annoying syntactical quirks from our experience with propositional logic.

First of all, just like in the case of propositional logic, formulas have variables. Let's give a preliminary definition (which should remind us of a definition we've seen before). This time it will take a bit longer to state.

Definition 1.2.1. Let $\mathcal{L}$ be a first-order language and t an $\mathcal{L}$-term. We define the *set of variables of* t, denoted $\mathsf{Var}(t)$, as follows:

(1) If t is the variable x then $\mathsf{Var}(t) = \{x\}$.

(2) If t is a constant symbol, then $\mathsf{Var}(t) = \emptyset$.

(3) If t is of the form $\underline{f}(t_1, \ldots, t_n)$ for some n-ary function symbol $\underline{f} \in \mathsf{Fun}(\mathcal{L})$ and $\mathcal{L}$-terms $t_1, \ldots, t_n$, then $\mathsf{Var}(t) = \bigcup_{i \leq n} \mathsf{Var}(t_i)$.

We say that t is *closed* if $\mathsf{Var}(t) = \emptyset$.

That's it with terms. Now, for formulas, as you may have expected, we have to start with the atomic guys and build from there.

Definition 1.2.2. Let ϕ be an atomic $\mathcal{L}$-formula. Then, the *set of variables of* ϕ, again denoted $\mathsf{Var}(\phi)$,[5] is defined as follows:

(1) If ϕ is of the form $t_1 \doteq t_2$ for $\mathcal{L}$-terms t_1, t_2 then $\mathsf{Var}(\phi) = \mathsf{Var}(t_1) \cup \mathsf{Var}(t_2)$.

(2) If ϕ is of the form $\underline{R}(t_1, \ldots, t_n)$ for some n-ary relation symbol $\underline{R} \in \mathsf{Rel}(\mathcal{L})$ and $\mathcal{L}$-terms $t_1, \ldots, t_n$, then $\mathsf{Var}(\phi) = \bigcup_{i \leq n} \mathsf{Var}(t_i)$.

Before we move up a level, to formulas proper, we need to make an important distinction. We want to be able to tell apart if a variables x in a formula appears in some quantifier, say $(\forall x)$ or $(\exists x)$ or not.

The following example could explain a bit what's going on here.

Example 1.2.3. In Calculus, you came across expressions of the form:

$$\int_a^b f(x)\mathrm{d}x.$$

Here are two integrals, which SYNTACTICALLY are different objects:

$$\int_a^b 3x^2 + 2x + \log(x)\mathrm{d}x, \text{ and } \int_a^b 3y^2 + 2y + \log(y)\mathrm{d}y.$$

Of course, any sane person understands that the objects these two integrals represent is the same – we are allowed to switch up the the variable x from the dx as long as we switch up in the same way all occurrences of x in our expression. A more murky example is the following:

$$f(y) = y \int y^2 \left(\int y^3 \mathrm{d}y + y \int \log(y)\mathrm{d}y \right) \mathrm{d}y.$$

[5] I am here overloading the notation Var we introduced earlier, but it should always be clear what I'm referring to. In fact, I overloaded that notation before as well, but I'm running out of meaningful symbols.

This is something a maniac would write. A more reasonable human being would write the above as follows:

$$f(x) = x \int y^2 \left(\int z^3 \mathrm{d}z + y \int \log(u) \mathrm{d}u \right) \mathrm{d}y.$$

I'm pretty sure you all can see what's happening here. Of course, this is only an aesthetic choice, but sometimes it's good to have good aesthetics.

In general, $(\forall x)$ and $(\exists x)$ in formulas act like $\mathrm{d}x$ in integrals, and we'd like to be able to make the distinctions that in integrals come to us rather naturally.

Definition 1.2.4. Let $\mathcal{L}$ be a first-order formula and ϕ an $\mathcal{L}$-formula. We define three things at the same time:

(1) The *variables of* ϕ, denoted $\mathsf{Var}(\phi)$;[6]

(2) The *free variables of* ϕ, denoted $\mathsf{Free}(\phi)$;

(3) The *bound variable of* ϕ, denoted $\mathsf{Bound}(\phi)$;

all, by induction on the structure of ϕ, as follows:

(1) If ϕ is atomic, then $\mathsf{Free}(\phi) = \mathsf{Var}(\phi)$,[7] $\mathsf{Bound}(\phi) = \emptyset$.

(2) Suppose we have defined our three sets for the $\mathcal{L}$-formulas ϕ_1 and ϕ_2.

- $\mathsf{Var}(\phi_1 \wedge \phi_2) = \mathsf{Var}(\phi_1) \cup \mathsf{Var}(\phi_2)$,
  $\mathsf{Free}(\phi_1 \wedge \phi_2) = \mathsf{Free}(\phi_1) \cup \mathsf{Free}(\phi_2)$,
  $\mathsf{Bound}(\phi_1 \wedge \phi_2) = \mathsf{Bound}(\phi_1) \cup \mathsf{Bound}(\phi_2)$;
- $\mathsf{Var}(\phi_1 \vee \phi_2) = \mathsf{Var}(\phi_1) \cup \mathsf{Var}(\phi_2)$,
  $\mathsf{Free}(\phi_1 \vee \phi_2) = \mathsf{Free}(\phi_1) \cup \mathsf{Free}(\phi_2)$,
  $\mathsf{Bound}(\phi_1 \vee \phi_2) = \mathsf{Bound}(\phi_1) \cup \mathsf{Bound}(\phi_2)$;
- $\mathsf{Var}(\phi_1 \to \phi_2) = \mathsf{Var}(\phi_1) \cup \mathsf{Var}(\phi_2)$,
  $\mathsf{Free}(\phi_1 \to \phi_2) = \mathsf{Free}(\phi_1) \cup \mathsf{Free}(\phi_2)$,
  $\mathsf{Bound}(\phi_1 \to \phi_2) = \mathsf{Bound}(\phi_1) \cup \mathsf{Bound}(\phi_2)$;
- $\mathsf{Var}(\neg\phi_1)) = \mathsf{Var}(\phi_1)$,
  $\mathsf{Free}(\neg\phi_1)) = \mathsf{Free}(\phi_1)$,
  $\mathsf{Bound}(\neg\phi_1)) = \mathsf{Bound}(\phi_1)$;

[6]You got it, I'm overloading, again.

[7]That is, in an atomic formula, ALL variables are free.

- And now we come to the heart of the cheese, $\mathsf{Var}((\forall x)\phi) = \mathsf{Var}(\phi)$, but
$$\mathsf{Free}((\forall x)\phi) = \mathsf{Free}(\phi) \setminus \{x\}.$$
and
$$\mathsf{Bound}((\forall x)\phi) = \begin{cases} \mathsf{Bound}(\phi) \cup \{x\}, & \text{if } x \in \mathsf{Free}(\phi) \\ \mathsf{Bound}(\phi), & \text{if } x \notin \mathsf{Free}(\phi) \end{cases}$$

We call an $\mathcal{L}$-formula ϕ an $\mathcal{L}$-*sentence* if $\mathsf{Free}(\phi) = \emptyset$.

Okay that was a total mess – I'm sorry. There are some somewhat subtle (and mostly silly) points that we have to worry about for a minute! Soon we will realise that we are sensible people and stop worrying about them, but such is life. Here's the first point:

A variable in a formula can be both bound and free.[8]

The second, less important point is the following:

Not every variable that appears next to a quantifier is bound.

The third and last annoyance we could have is the following:

Quantifiers could bind the same variables.

That last annoyance may not mean much to you right now, but as sensible people, we'd really hope to avoid this. Let's try to illustrate this with some examples:

Example 1.2.5. I'll not worry too much about $\mathcal{L}$ here, that's not the point. Suppose that $\mathcal{L}$ has a single relation symbol $\underline{R}$ of arity 2.

(1) The formula $\underline{R}(x, y)$ has two variables, x and y and they are both free.

(2) The formula $\underline{R}(x, y) \wedge (\exists x)\underline{R}(x, y)$ has two variables x and y, two free variables x and y and one bound variable x.

(3) The formula $\underline{R}(x, y) \wedge (\forall z)\underline{R}(x, y)$ has two variables x and y, two free variables x and y and no bound variables.

(4) The formula $(\forall x)(\forall y)(x \doteq y)$ is a sentence.

(5) The formula $\underline{R}(x, x) \vee (\forall x)((\forall x)\underline{R}(x, x) \to \underline{R}(x, x))$ is a mess.

This is all very annoying. We can't really get rid of the second point, so we'll define it away:

[8]That is to say it's not always the case that $\mathsf{Bound}(\phi) \cap \mathsf{Free}(\phi) \neq \emptyset$.

Definition 1.2.6. Let $\mathcal{L}$ be a first-order language and ϕ an $\mathcal{L}$-formula. If ϕ is of the form $(\forall x)\psi$ (respectively $(\exists x)\psi$) and $x \notin \mathsf{Bound}(\phi)$ (i.e. $x \notin \mathsf{Free}(\psi)$), then we say that the quantifier $(\forall x)$ (resp. $(\exists x)$) is *vacuous* in ϕ.

In the next subsubsubsection, we'll do a lot of syntactical yoga to get rid of the first and third annoyances. The issue is that formalising common sense can sometimes be hard. That being said, the contents of this subsubsection are not really required for the remainder of this chapter, but it's there as a warm-up for harder substitution procedures to come. Read at your peril.

1.2.1. *Doing the obvious can be hard.* The first part of the next definition should ring some bells:

Definition 1.2.7. Let $\mathcal{L}$ be a first-order language, t an $\mathcal{L}$-term and $x, y \in \mathsf{Var}$. We define the term $t[y/x]$ inductively, as follows:

(1) If t is the variable x then $t[y/x]$ is the variable y.

(2) If t is a constant symbol, then $t[y/x]$ is just t.

(3) If t is of the form $\underline{f}(t_1, \dots, t_n)$ for some n-ary function symbol $\underline{f} \in \mathsf{Fun}(\mathcal{L})$ and $\mathcal{L}$-terms $t_1, \dots, t_n$, then $t[y/x]$ is the term $\underline{f}\,(t_1[y/x], \dots, t_n[y/x])$.

That's it for terms, lets go to atomic formulas:

Definition 1.2.8. Let $\mathcal{L}$ be a first-order language, ϕ an atomic $\mathcal{L}$-formula and $x, y \in \mathsf{Var}$. Then, the atomic formula $\phi[y/x]$ is defined inductively as follows

(1) If ϕ is of the form $t_1 \doteq t_2$ for $\mathcal{L}$-terms t_1, t_2 then $\phi[y/x]$ is the formula $t_1[y/x] \doteq t_2[y/x]$.

(2) If ϕ is of the form $\underline{R}(t_1, \dots, t_n)$ for some n-ary relation symbol $\underline{R} \in \mathsf{Rel}(\mathcal{L})$ and $\mathcal{L}$-terms $t_1, \dots, t_n$, then $\phi[y/x]$ is the formula $\underline{R}(t_1[y/x], \dots, t_n[y/x])$.

We now have the tools to define a notion of *"free-variable" substitution*. The definition is somewhat complicated but let's try to bare through it together:

Definition 1.2.9. Let $\mathcal{L}$ be a first-order language, ϕ an $\mathcal{L}$-formula and $x, y \in \mathsf{Var}$. We define $\phi[y/x]_{free}$ inductively, as follows:

(1) If ϕ is atomic, then $\phi[y/x]_{free}$ is just the atomic formula $\phi[y/x]$ we defined above.

(2) Suppose we have defined $\phi_1[y/x]_{free}$ and $\phi_2[y/x]_{free}$. Then:

- $(\phi_1 \wedge \phi_2)[y/x]_{free}$ is the formula $(\phi_1[y/x]_{free} \wedge \phi_2[y/x]_{free})$.
- $(\phi_1 \vee \phi_2)[y/x]_{free}$ is the formula $(\phi_1[y/x]_{free} \vee \phi_2[y/x]_{free})$.
- $(\phi_1 \to \phi_2)[y/x]$ is the formula $(\phi_1[y/x] \to \phi_2[y/x])$.
- $(\neg\phi_1)[y/x]_{free}$ is the formula $(\neg\phi_1[y/x]_{free})$.
- $(\forall z)\phi[y/x]_{free}$ is defined as follows:
$$(\forall z)\phi[y/x]_{free} := \begin{cases} (\forall x)\phi & \text{if } z = x \\ (\forall z)(\phi[y/x]_{free}) & \text{otherwise.} \end{cases}$$
- $(\exists z)\phi[y/x]_{free}$ is similarly defined as follows:
$$(\exists z)\phi[y/x]_{free} := \begin{cases} (\exists x)\phi & \text{if } z = x \\ (\exists z)(\phi[y/x]_{free}) & \text{otherwise.} \end{cases}$$

Intuitively, to compute $\phi[y/x]_{free}$ from ϕ we go down the construction of ϕ and whenever we are not inside some quantifier of the form $(\forall x)$ or $(\exists x)$ (for the SAME x), we change every occurrence of x with y. This is all annoying, but we have to do it if we're serious about things. Let's look at some examples:

Example 1.2.10. Continuing with our binary language from the previous example, say ϕ is the formula:
$$(\forall z)((\forall x)R(x,z) \vee R(x,x)).$$
Then, $(\forall z)((\forall x)R(x,z) \vee R(x,x))[y/x]_{free}$ is computed as follows:

Step 1. $(\forall z)((\forall x)R(x,z) \vee R(x,x))[y/x]_{free} = (\forall z)\big(((\forall x)R(x,z) \vee R(x,x))[y/x]_{free}\big)$.

Step 2. $((\forall x)R(x,z) \vee R(x,x))[y/x]_{free} = ((\forall x)R(x,z)[y/x]_{free} \vee R(x,x)[y/x]_{free})$

Step 3. $(\forall x)R(x,z)[y/x]_{free} = (\forall x)R(x,z)$ and $R(x,x)[y/x]_{free} = R(y,y)$.

Step 4. Putting everything together we have:
$$(\forall z)((\forall x)R(x,z) \vee R(x,x))[y/x]_{free} = (\forall z)((\forall x)R(x,z) \vee R(y,y)),$$

So, all the free occurrences of x were replaced by y. On the other hand,

On the other hand, consider the formula:
$$(\forall z)(\forall x)R(x,z)$$
Carrying out the analogous steps, we see that nothing will happen if we apply the free-occurrence substitution we defined above, since there are no free occurrences of x in this formula.

Just like in propositional logic, we can inductively define

$$\phi[y_1/x_1, \ldots, y_n/x_n]_{free}.$$

Exercise 1.2.11. To make sure you understand what happened above, write out the proper inductive definition of $\phi[y_1/x_1, \ldots, y_n/x_n]_{free}$.

Exercise 1.2.12. Let $\mathcal{L}$ be a first-order language and ϕ an $\mathcal{L}$-formula of height h. Prove that for any variables $x_1, \ldots, x_n, y_1, \ldots, y_n \in \mathsf{Var}$, the formula $\phi[y_1/x_1, \ldots, y_n/x_n]_{free}$ is an $\mathcal{L}$-formula of height n, and has the same structure as ϕ.

All of this for the following definition:

Definition 1.2.13. We say that an $\mathcal{L}$-formula ϕ is *clean* if:

(1) $\mathsf{Bound}(\phi) \cap \mathsf{Free}(\phi) = \emptyset$.
(2) For any $x \in \mathsf{Var}$, if ϕ has a subformula of the form $(\forall x)\psi$ or $(\exists x)\psi$, then $x \notin \mathsf{Bound}(\psi)$.

Thus, a formula ϕ is clean if and only if all of its free and bound variables are distinct, and if $(\forall x)\psi$ is a **subformula**[9] of ϕ, then x is not a bound variable of a subformula of ψ.

The method to clean up formulas may seem rather convoluted, but I promise that once you get the hang of it, it will become second nature:

Definition 1.2.14. Given an $\mathcal{L}$-formula ϕ, we define $\mathsf{pre\text{-}clean}(\phi)$, by induction, as follows:

(1) If ϕ is atomic, then $\mathsf{pre\text{-}clean}(\phi) = \phi$.
(2) Suppose that ϕ is a formula of height $n+1$ and we have defined $\mathsf{pre\text{-}clean}(\psi)$ for all formulas of height n. We define $\mathsf{pre\text{-}clean}(\phi)$ as follows:
    - If ϕ is of the form $(\phi_1 \wedge \phi_2)$, then $\mathsf{pre\text{-}clean}(\phi)$ is $(\mathsf{pre\text{-}clean}(\phi_1) \wedge \mathsf{pre\text{-}clean}(\phi_2))$.
    - When ϕ is of the form $(\phi_1 \vee \phi_2)$, $(\phi_1 \to \phi_2)$ or $(\neg \phi_1)$, we define $\mathsf{pre\text{-}clean}(\phi)$ similarly.

[9]The subformulas of a first-order formula are defined analogously to the subformulas of a propositional formula.

- If ϕ is of the form $(\forall x)\phi_1$, then $\mathsf{pre\text{-}clean}(\phi)$ is defined to be $(\forall y)\mathsf{pre\text{-}clean}(\phi_1[y/x]_{free})$ for some $y \notin \mathsf{Var}(\phi_1)$.
- If ϕ is of the form $(\exists x)\phi_1$, then $\mathsf{pre\text{-}clean}((\exists x)\phi_1)$ is defined to be $(\exists y)\mathsf{pre\text{-}clean}(\phi_1[y/x]_{free})$ for some $y \notin \mathsf{Var}(\phi_1)$.

Once we have defined $\mathsf{pre\text{-}clean}(\phi)$ for any $\mathcal{L}$-formula ϕ, we define $\mathsf{clean}(\phi)$ to be the formula:

$$(\mathsf{pre\text{-}clean}(\phi))[y_1/x_1, \ldots, y_m/x_m]_{free},$$

where $\{x_1, \ldots, x_m\} = \mathsf{Free}(\mathsf{pre\text{-}clean}(\phi))$, and $\{y_1, \ldots, y_m\} \in \mathsf{Var} \setminus \mathsf{Var}(\mathsf{pre\text{-}clean}(\phi))$.

Let's make our lives really hard for a minute:

Example 1.2.15. Consider the sentence ϕ in a language with a single relation symbol $\underline{R}$ of arity 1 (these fellas are sometimes called, in increasing level's of fanciness: **predicates** or **unary relations** or **monadic relations**):[10]

$$(\forall x)\Bigg((\forall x)\Big((\forall x)R(x) \to R(x)\Big) \to R(x)\Bigg)$$

We compute $\mathsf{pre\text{-}clean}(\phi)$:

Step 1. By definition, $\mathsf{pre\text{-}clean}\Bigg((\forall x)\Bigg((\forall x)\Big((\forall x)R(x) \to R(x)\Big) \to R(x)\Bigg)\Bigg)$ is:

$$(\forall y)\mathsf{pre\text{-}clean}\Bigg(\Bigg((\forall x)\Big((\forall x)R(x) \to R(x)\Big) \to R(x)\Bigg)[y/x]_{free}\Bigg)$$

Step 2. Computing the free-variable substitution above we get:

$$(\forall y)\mathsf{pre\text{-}clean}\Big((\forall x)\Big((\forall x)R(x) \to R(x)\Big) \to R(y)\Big)$$

Step 3. We repeat Step 1, for the formula $(\forall x)\Big((\forall x)R(x) \to R(x)\Big) \to R(y)\Big)$ to get:

$$(\forall z)\Big((\mathsf{pre\text{-}clean}((\forall x)R(x)) \to R(z)\Big) \to \mathsf{pre\text{-}clean}(R(y))\Big)$$

Step 4. After we start having some faith, we see that once we put everything back together we get the following clean formula:

$$(\forall y)\Bigg((\forall z)\Big((\forall u)R(u) \to R(y)\Big) \to R(z)\Bigg)$$

[10]This may have some unnecessary brackets as we go down the steps, but I'm putting these just to make the reading a bit cleaner.

Exercise 1.2.16. Write out in detail the missing cases from the definition of $\mathsf{pre\text{-}clean}$.

To get some practice with the definitions, you are welcome to do the following annoying exercise in induction:

Exercise 1.2.17. Let ϕ be an $\mathcal{L}$-formula. Show that:

(1) ϕ and $\mathsf{pre\text{-}clean}(\phi)$ have the same structure.

(2) $\mathsf{Free}(\phi) = \mathsf{Free}(\mathsf{pre\text{-}clean}(\phi))$.

(3) $\mathsf{Bound}(\phi) \cap \mathsf{Bound}(\mathsf{pre\text{-}clean}(\phi)) = \emptyset$.

(4) $\mathsf{Bound}(\mathsf{pre\text{-}clean}(\phi)) = \mathsf{Bound}(\mathsf{clean}(\phi))$.

Now, for the main course:

Lemma 1.2.18. *For any first-order language $\mathcal{L}$ and any $\mathcal{L}$-formula ϕ, the $\mathcal{L}$-formula $\mathsf{clean}(\phi)$ is a clean formula.*

PROOF. We have to show two things:

(1) For all $x \in \mathsf{Var}(\mathsf{clean})(\phi)$, if $x \in \mathsf{Free}(\mathsf{clean}(\phi))$, then $x \notin \mathsf{Bound}(\mathsf{clean}(\phi))$. Let $x \in \mathsf{Var}(\mathsf{clean}(\phi))$. By definition, we have that:

$$\mathsf{clean}(\phi) = (\mathsf{pre\text{-}clean}(\phi))[y_1/x_1, \ldots, y_m/x_m]_{free},$$

where $\{x_1, \ldots, x_m\} = \mathsf{Free}(\mathsf{pre\text{-}clean}(\phi))$, and $\{y_1, \ldots, y_m\} \in \mathsf{Var}\backslash\mathsf{Var}(\mathsf{pre\text{-}clean}(\phi))$. Thus, if $x \in \mathsf{Free}(\mathsf{clean}(\phi))$, we have that $x = y_i$, for some $i \leq m$. But then, $x \notin \mathsf{Var}(\mathsf{pre\text{-}clean}(\phi))$ and, in particular, $x \notin \mathsf{Bound}(\mathsf{pre\text{-}clean}(\phi)) = \mathsf{Bound}(\mathsf{clean}(\phi))$, by the previous exercise.

(2) Now, to show that if $\mathsf{clean}(\phi)$ has a subformula of the form $(\forall x)\psi$ (or $(\exists x)\psi$), then $x \notin \mathsf{Bound}(\psi)$. It suffices to show this when $\mathsf{clean}(\phi)$ is itself of the form $(\forall x)\psi$ (see next exercise). By definition, we know that if $\mathsf{clean}(\phi)$ is of this form, then $\mathsf{clean}(\phi)$ is the formula:

$$(\forall x)(\mathsf{pre\text{-}clean}(\chi)[y_1/x_1, \ldots, y_m/x_m]_{free}),$$

for some formula χ, where $x \notin \{x_1, \ldots, x_n\}$. By the previous exercise:

$$\begin{aligned}\mathsf{Bound}(\psi) &= \mathsf{Bound}(\mathsf{clean}(\chi)) \\ &= \mathsf{Bound}(\mathsf{pre\text{-}clean}(\chi)[y_1/x_1, \ldots, y_m/x_m]_{free})) \\ &\subseteq \mathsf{Var}(\chi)\end{aligned}$$

so, if $x \in \mathsf{Bound}(\psi)$, then $x \in \mathsf{Var}(\chi)$, which is a contradiction.

□

Exercise 1.2.19. Write down the details of the proof of the lemma above.

From now on, we will assume ALL FORMULAS we are dealing with are clean, by implicitly always replacing a given formula ϕ with $\mathsf{clean}(\phi)$. When we discuss semantics, we will see that this is purely an aesthetic assumption.

End of digression.

1.2.2. *Another approach to cleanliness.* All of this was very annoying, and in fact, just a formal way of doing something very trivial. Given an $\mathcal{L}$-formula ϕ of the form $(\forall x)\psi$, we say that the **scope** of the quantifier $(\forall x)$ is the subformula ψ. Any *free* occurrence of the variable x in ψ is bounded by the quantifier $(\forall x)$. If somewhere in ψ there is a subformula of the form $(\forall x)\chi$ (or $(\exists x)\chi$), then it is *this* quantifier that binds all the occurrences of x in χ. For example, in the formula that we were playing with before:

$$(\forall x)\Big((\forall x)\big((\forall x)R(x) \to R(x)\big) \to R(x)\Big)$$

the scope of the outermost quantifier is:

$$(\forall x)\color{blue}{\Big((\forall x)\big((\forall x)R(x) \to R(x)\big) \to R(x)\Big)}$$

so all FREE occurrences of x in the blue part of the formula are bound by this outer quantifier. Similarly, we can see that the scope of the second quantifier is:

$$(\forall x)\Big((\forall x)\color{red}{\big((\forall x)R(x) \to R(x)\big)} \to R(x)\Big)$$

so again, the FREE occurrences of x in the red part are bound by this intermediate quantifier. Finally, the scope of the innermost quantifier is:

$$(\forall x)\Big((\forall x)\big((\forall x)\color{green}{R(x)} \to R(x)\big) \to R(x)\Big).$$

All in all, the occurrences of x bound by each quantifier are as follows:

$$\color{blue}{(\forall x)}\Big(\color{red}{(\forall x)}\big(\color{green}{(\forall x)}R(\color{green}{x}) \to R(\color{red}{x})\big) \to R(\color{blue}{x})\Big).$$

(Pre-)Cleaning up the formula simply involved giving different names to the variables in each colour. The final clean up step involves renaming all free variables so that they don't clash with the quantified variables. Now that we no longer have to worry about our formulas being dirty, let's set up some (nice, for a change) notation.

Notation 1.2.20. Let ϕ be an $\mathcal{L}$-formula. We will write $\phi(x_1, \ldots, x_n)$ to indicate the following two things:

(1) $\mathsf{Free}(\phi) \subseteq \{x_1, \ldots, x_n\}$.

(2) $x_i \neq x_j$ whenever $1 \leq i \neq j \leq n$.

2. S⁀e⁀m⁀a⁀n⁀t⁀i⁀c⁀s

2.1. Structures and assignments. Okay enough syntax for a minute, let's take a breath, and talk about what our formulas mean. In propositional logic, our sentences were always either true or false, but to make our formulas make sense, we needed to give each propositional variable a meaning, a so called assignment to either true or false. So the "place" in which propositional formulas take meaning is some universe that tells each variable if it should be T or F. Here, the goal is not to describe reasoning, but to describe mathematics, so kinda necessarily whatever this "universe" is, it will have to be a somewhat more complicated one.

Definition 2.1.1. Let $\mathcal{L}$ be a first-order language. An *$\mathcal{L}$-structure* $\mathcal{M}$ consist of a non-empty set M together with:

(1) An element $c^{\mathcal{M}} \in M$, for each constant symbol $\underline{c} \in \mathsf{Const}(\mathcal{L})$

(2) A subset $R^{\mathcal{M}} \subseteq M^n$, for each n-ary relation symbol $\underline{R} \in \mathsf{Rel}(R)$.

(3) A function $f^{\mathcal{M}} : M^n \to M$, for each n-ary function symbol $\underline{f} \in \mathsf{Fun}(\mathcal{L})$.

Altogether, we often write:

$$\mathcal{M} = \Big(M; \left(c^{\mathcal{M}}\right)_{\underline{c} \in \mathsf{Const}(\mathcal{L})}, \left(R^{\mathcal{M}}\right)_{\underline{R} \in \mathsf{Rel}(\mathcal{L})}, \left(f^{\mathcal{M}}\right)_{\underline{f} \in \mathsf{Fun}(\mathcal{L})} \Big).$$

We call M the *domain* or *universe* of the $\mathcal{L}$-structure $\mathcal{M}$. For each constant symbol $\underline{c} \in \mathsf{Const}(\mathcal{L})$ we call $c^{\mathcal{M}}$ the *interpretation* of $\underline{c}$ in $\mathcal{M}$ and similarly for relation symbols and function symbols.

Sometimes, we'll work with a language $\mathcal{L}$ we intuitively understand, such as $\mathcal{L}_{Peano}$, but of course in an $\mathcal{L}$-structure the interpretations of the symbols of the language can be as insane as we want. Let's do some examples to make sure we're on the same page:

Example 2.1.2. A pretty standard (this will be the punchline to a joke whose set up will appear several chapters later) $\mathcal{L}_{Peano}$-structure, $\mathcal{N}_{st}$ is the following:

- The universe of $\mathcal{N}_{st}$ is $\mathbb{N}$, the set of all natural numbers.
- The interpretation of the constant symbol $\underline{0}$ is the natural number 0.
- The interpretation of the unary function symbol $\underline{S}$ is the **successor function**:
$$\begin{aligned} \mathsf{succ} : \mathbb{N} &\to \mathbb{N} \\ x &\mapsto x+1 \end{aligned}$$
- The interpretation of the binary function symbols $\underline{+}$ and $\underline{\times}$ are the usual addition and multiplication functions, respectively.

Another example of an $\mathcal{L}_{Peano}$-structure, $\mathcal{N}_{looney}$ is the following:

- The universe of $\mathcal{N}_{looney}$ is $\omega^+ = \mathbb{N} \cup \{\mathbb{N}\}$, the successor ordinal of ω.
- The interpretation of the constant symbol $\underline{0}$ is the natural number 3.
- The interpretation of the unary function symbol $\underline{S}$ is a "loopy" function:
$$\begin{aligned} \mathsf{succ} : \omega^+ &\to \omega^+ \\ x &\mapsto \begin{cases} x+1 & \text{if } x \in \mathbb{N} \\ 0 & \text{if } x = \mathbb{N}. \end{cases} \end{aligned}$$
- The interpretation of the binary function symbols $\underline{+}$ and $\underline{\times}$ are constant functions that always return the number 9.

Both of these are very valid $\mathcal{L}_{Peano}$-structures.

Now, we know all about writing formulas, so given an $\mathcal{L}$-formula $\phi(x_1, \dots, x_n)$ (remember, this means that ϕ is allowed to have free variables, and if it does, these are amongst $x_1, \dots, x_n$) and an $\mathcal{L}$-structure $\mathcal{M} = (M; \dots)$, we want to start understanding what ϕ could mean in $\mathcal{M}$. Of course, the meaning of ϕ should (and will) depend on the values that its variables take in M, just like in the case of propositional logic.

The next definition is, again, something we have seen before, in the context of propositional logic:

Definition 2.1.3. Let $\mathcal{L}$ be a first-order language and $\mathcal{M} = (M; \dots)$ an $\mathcal{L}$-structure. An *assignment* is just a function α that assigns to each variable $x \in \mathsf{Var}$ an element of M, i.e. a function $\alpha : \mathsf{Var} \to M$.

So, for example, if the domain of an $\mathcal{L}$-structure is $\mathbb{N}$, and $\mathsf{Var} = \{x_0, x_1, \dots\}$ (this is always possible, since we've assumed that our variables are a countable set, the use of other letters for variables is just to make things look pretty), then the following function is an assignment:

$$\begin{aligned}\alpha : \mathsf{Var} &\to \mathbb{N}\\ x_i &\mapsto i\end{aligned}$$

I could almost give you, now, the definition of the semantics of first-order formulas, but first I'll need one more little notion. Given an assignment, we will define an important way of adjusting it, namely changing the value it gives to a single variable:

Notation 2.1.4. Let $\alpha : \mathsf{Var} \to M$ be an assignment, $x \in \mathsf{Var}$ and $b \in M$. We write $\alpha_{b/x}$ for the assignment:

$$\begin{aligned}\alpha_{b/x} : \mathsf{Var} &\to M\\ y &\mapsto \begin{cases} b & \text{if } x = y\\ \alpha(y) & \text{otherwise.}\end{cases}\end{aligned}$$

More generally, let $x_1, \dots, x_n \in \mathsf{Var}$ be pairwise distinct variables and $b_1, \dots, b_n \in M$. Then we define:

$$\alpha_{b_1/x_1,\dots,b_n/x_n} := (\alpha_{b_1/x_1,\dots,b_{n-1}/x_{n-1}})_{b_n/x_n}.$$

This is another one of those inductive definitions. We change the values α gives to x_i, one at a time.

Great, that was abstract and seemingly useless. I feel like the best thing to do now is not to give you an example or whatever, but rather to keep throwing abstract definitions at you.

Definition 2.1.5. Let $\mathcal{L}$ be a first-order language, $\mathcal{M} = (M; \dots)$ an $\mathcal{L}$-structure, and t an $\mathcal{L}$-term. Given an assignment $\alpha : \mathsf{Var} \to \mathcal{M}$, we define the *interpretation of t in $\mathcal{M}$ under α*, denoted $t^{\mathcal{M}}[\alpha]$, by induction, as follows:

(1) If t is the variable x, then $t^{\mathcal{M}}[\alpha]$ is the element $\alpha(x)$ of M.

(2) If t is the constant symbol $\underline{c}$, then $t^{\mathcal{M}}[\alpha]$ is the element $c^{\mathcal{M}}$ of M.

(3) If t is of the form $\underline{f}(t_1, \dots, t_n)$, for an n-ary function symbol $f \in \mathsf{Fun}(\mathcal{L})$ and $\mathcal{L}$-terms $t_1, \dots, t_n$, then $t^{\mathcal{M}}[\alpha]$ is the element $f^{\mathcal{M}}(t_1^{\mathcal{M}}[\alpha], \dots, t_n^{\mathcal{M}}[\alpha])$ of M.

Example 2.1.6. Recall our $\mathcal{L}_{Peano}$-structure $\mathcal{N}_{st}$ from before, and the assignment $\alpha : \mathsf{Var} \to \mathbb{N}$. The interpretation under α of the term $x_1 \underline{\times} x_2$ in $\mathcal{N}_{st}$ is just the natural number 2.

To make life a little bit more complicated, recall our other $\mathcal{L}_{Peano}$-structure, $\mathcal{N}_{looney}$. An interpretation for this structure is just a map $\mathsf{Var} \to \omega^+$, and since Var is countable, we can just assume that $\mathsf{Var} = \{x_1, x_2, \dots\} \cup \{x_\omega\}$. Take α to be the function that maps x_i to i (where i is allowed to be $\mathbb{N}$). Then, the interpretation of $x_1 \underline{\times} x_\omega$ is the natural number 9. Remember, $\underline{\times}$ in this structure was the constant function that always returned the number 9. The interpretation of $\underline{S}(\underline{0})$ is the natural number 10, and the interpretation of $\underline{S}(x_\omega)$ is the natural number 0.

Here's a lemma that should also look familiar:

Lemma 2.1.7. *Let $\mathcal{L}$ be a first-order language, $\mathcal{M}$ an $\mathcal{L}$-structure and t an $\mathcal{L}$-term. Let α and β be two assignments such that for all $x \in \mathsf{Var}(t)$ we have $\alpha(x) = \beta(x)$. Then, $t^{\mathcal{M}}[\alpha] = t^{\mathcal{M}}[\beta]$.*

Proof. We prove this by induction on the length of terms:

- If t is the variable x, then, by assumption we have that $\alpha(x) = \beta(x)$, and thus $t^{\mathcal{M}}[\alpha] = \alpha(x) = \beta(x) = t^{\mathcal{M}}[\beta]$.
- If t is a constant symbol $\underline{c}$ then $t^{\mathcal{M}}[\alpha] = c^{\mathcal{M}} = t^{\mathcal{M}}[\beta]$.
- Suppose that t is of the form $\underline{f}(t_1, \dots, t_n)$ for some n-ary function symbol $\underline{f}$ and $\mathcal{L}$-terms $t_1, \dots, t_n$. Given two assignments $\alpha, \beta : \mathsf{Var} \to \mathcal{M}$, if they agree on all the variables in t, then they agree on all the variables of each t_i, for $i \leq n$. Thus, by induction $t_i^{\mathcal{M}}[\alpha] = t_i^{\mathcal{M}}[\beta]$, for $i \leq n$. So:
$$t^{\mathcal{M}}[\alpha] = f^{\mathcal{M}}(t_1^{\mathcal{M}}[\alpha], \dots, t_n^{\mathcal{M}}[\alpha]) = f^{\mathcal{M}}(t_1^{\mathcal{M}}[\beta], \dots, t_n^{\mathcal{M}}[\beta]) = t^{\mathcal{M}}[\beta],$$
and we're done. $\square$

We will now extend Notation 1.2.20 to terms, in a natural way:

Notation 2.1.8. If t is an $\mathcal{L}$-term with variables amongst $x_1, \dots, x_n$, then we may denote this by $t(x_1, \dots, x_n)$.[11]

2.2. Truth.

We have seen how to use assignments to give meaning to terms. Let's put all of this together, to define what it means for a formula to be true in a structure.

Definition 2.2.1 (Tarski's definition of truth)**.** Let $\mathcal{L}$ be a first-order language, $\mathcal{M} = (M; \dots)$ an $\mathcal{L}$-structure, and ϕ be an $\mathcal{L}$-formula. Let $\alpha : \mathsf{Var} \to M$ be an

[11]Recall that all variables in a term are free.

assignment. We define what it means for ϕ *to be satisfied in* $\mathcal{M}$*, under the assignment* α, denoted $\mathcal{M} \vDash \phi[\alpha]$, by induction on the structure of ϕ, as follows:

(1) If ϕ is an atomic formula, then there are two cases to consider:

- $\mathcal{M} \vDash (t_1 \doteq t_2)[\alpha]$ if and only if $t_1^{\mathcal{M}}[\alpha] = t_2^{\mathcal{M}}[\alpha]$, for $\mathcal{L}$-terms t_1 and t_2
- $\mathcal{M} \vDash \underline{R}(t_1, \dots, t_n)[\alpha]$ if and only if $R^{\mathcal{M}}(t_1^{\mathcal{M}}[\alpha], \dots, t_n^{\mathcal{M}}[\alpha])$, for $\underline{R} \in \mathsf{Rel}(\mathcal{L})$ an n-ary relation symbol and $\mathcal{L}$-terms $t_1, \dots, t_n$.

(2) If we have defined $\mathcal{M} \vDash \phi_1[\alpha]$ and $\mathcal{M} \vDash \phi_2[\alpha]$, then we set:

(a) $\mathcal{M} \vDash (\phi_1 \wedge \phi_2)[\alpha]$ if and only if $\mathcal{M} \vDash \phi_1[\alpha]$ and $\mathcal{M} \vDash \phi_2[\alpha]$.

(b) $\mathcal{M} \vDash (\phi_1 \vee \phi_2)[\alpha]$ if and only if $\mathcal{M} \vDash \phi_1[\alpha]$ or $\mathcal{M} \vDash \phi_2[\alpha]$.

(c) $\mathcal{M} \vDash (\neg\phi_1)[\alpha]$ if and only if it is not the case that $\mathcal{M} \vDash \phi[\alpha]$.

(d) $\mathcal{M} \vDash (\forall x)\phi[\alpha]$ if and only if for all $b \in M$ we have that $\mathcal{M} \vDash \phi_1[\alpha_{b/x}]$.

(e) $\mathcal{M} \vDash (\exists x)\phi[\alpha]$ if and only if for some $b \in M$ we have that $\mathcal{M} \vDash \phi_1[\alpha_{b/x}]$.

We write $\mathcal{M} \nvDash \phi_1[\alpha]$ as shorthand for: "it is not the case that $\mathcal{M} \vDash \phi[\alpha]$".[12]

This definition seems rather void of content. But it's really the core of this whole thing. The content of the definition is that the syntax we came up to express stuff has the meaning we want it to have. In all of the above cases, we are giving real meaning to a purely syntactic relation.

Example 2.2.2. Here's a very silly example. You'll be asked to do more examples, most of them less silly in HW3. In our structure $\mathcal{N}_{looney}$, for any assignment α, we have that:

$$\mathcal{N}_{looney} \vDash x_1 \underline{\times} x_2 \doteq \underline{S}(\underline{S}(\underline{S}(\underline{S}(\underline{S}(\underline{S}(\underline{0}))))))[\alpha].$$

What assignments do in $\mathcal{N}_{st}$ is much more interesting and complicated.

Just like when we did truth tables the truth tables themselves didn't say much, this is the grown-up version of truth tables (indeed, the first four clauses are precisely the truth tables from propositional logic). Let's do some logic!

Lemma 2.2.3. *Let* $\mathcal{M}$ *be an* $\mathcal{L}$*-structure and* ϕ *an* $\mathcal{L}$*-formula. Then, the following are equivalent:*

(1) $\mathcal{M} \vDash \phi[\alpha]$

[12]Immediately from the definition, we have that $\mathcal{M} \vDash \neg\phi[\alpha]$ if and only if $\mathcal{M} \nvDash \phi[\alpha]$.

(2) $\mathcal{M} \nvDash \neg\phi[\alpha]$

(3) $\mathcal{M} \vDash \neg\neg\phi[\alpha]$

for any assignment α*.*

This will be a bit of a word-salad, but it does have some moral point, it tells us that for any assignment α we can replace the syntactic object ϕ by the syntactic object $\neg\neg\phi$, and the semantic notions of satisfaction will be unaffected, which (unless you're an intuit) is a great thing.

Proof. The equivalence of (2) and (3) is by (the footnote in the) definition. We show that (1) and (2) are equivalent.

We have to prove this by induction on the structure of ϕ. Suppose that ϕ is atomic. Then:

(1) Suppose that ϕ is of the form $t_1 \doteq t_2$ for $\mathcal{L}$-terms t_1 and t_2. Then:

$$\begin{aligned}\mathcal{M} \vDash (t_1 \doteq t_2)[\alpha] &\text{ iff } t_1^{\mathcal{M}}[\alpha] = t_2^{\mathcal{M}}[\alpha]\\ &\text{iff it is not the case that } t_1^{\mathcal{M}}[\alpha] \neq t_2^{\mathcal{M}}[\alpha]\\ &\text{iff it is not the case that it is not the case that } t_1^{\mathcal{M}}[\alpha] = t_2^{\mathcal{M}}[\alpha]\\ &\text{if and only if it is not the case that } \mathcal{M} \vDash \neg(t_1 \doteq t_2)[\alpha]\\ &\text{iff } \mathcal{M} \nvDash \neg(t_1 \doteq t_2)[\alpha].\end{aligned}$$

(2) Suppose that ϕ is of the form $\underline{R}(t_1, \dots, t_n)$ for an n-ary relation symbol $\underline{R} \in \mathsf{Rel}(\mathcal{L})$ and $\mathcal{L}$-terms $t_1, \dots, t_n$. Then:

$$\begin{aligned}\mathcal{M} \vDash \underline{R}(t_1, \dots, t_n)[\alpha] &\text{ iff } R^{\mathcal{M}}(t_1^{\mathcal{M}}[\alpha], \dots, t_n^{\mathcal{M}}[\alpha])\\ &\text{iff it's not the case that it's not the case that } R^{\mathcal{M}}(t_1^{\mathcal{M}}[\alpha], \dots, t_n^{\mathcal{M}}[\alpha])\\ &\text{iff } \mathcal{M} \nvDash \neg\underline{R}(t_1, \dots, t_n)[\alpha].\end{aligned}$$

Now, for the inductive part. Suppose that we have shown the result for formulas ϕ_1 and ϕ_2 (and any assignments).

Exercise 2.2.4. Show that if $\mathcal{M} \vDash (\phi_1 \wedge \phi_2)[\alpha]$ if and only if $\mathcal{M} \nvDash (\neg(\phi_1 \wedge \phi_2))[\alpha]$. Then, do the analogous things for $\vee$, $\rightarrow$ and $\neg$.

Suppose that ϕ is of the form $(\forall x)\phi_1$. Then:

$\mathcal{M} \vDash (\forall x)\phi_1[\alpha]$ iff for all $b \in M$ we have $\mathcal{M} \vDash \phi_1[\alpha_{b/x}]$
iff for all $b \in M$ we have $\mathcal{M} \nvDash \neg\phi_1[\alpha_{b/x}]$
iff for all $b \in M$ it is not the case that $\mathcal{M} \vDash \neg\phi_1[\alpha_{b/x}]$
iff it's not the case that for some $b \in M$ we have $\mathcal{M} \vDash \neg\phi_1[\alpha_{b/x}]$
iff it's not the case that for some $b \in M$ it is not the case that $\mathcal{M} \vDash \phi_1[\alpha_{b/x}]$
iff it's not the case that it's not the case that for all $b \in M$ we have $\mathcal{M} \vDash \phi_1[\alpha_{b/x}]$
iff it's not the case that $\mathcal{M} \nvDash (\forall x)\phi_1$
iff it's not the case that $\mathcal{M} \vDash \neg(\forall x)\phi_1$
iff $\mathcal{M} \nvDash \neg(\forall x)\phi$,

which is what we wanted to show.

Exercise 2.2.5. Do the same when ϕ is of the form $(\exists x)\phi_1$.

Having done all the cases, the result follows. □

I mean it probably shouldn't feel like it not the case that something happened, or it should. Anyway it should feel like nothing happened. Here's more nothing:

THEOREM 2.2.6. *Let $\mathcal{M}$ be an $\mathcal{L}$-structure and ϕ an $\mathcal{L}$-formula. Then, the following are equivalent:*

(1) $\mathcal{M} \vDash (\forall x)\phi[\alpha]$

(2) $\mathcal{M} \vDash \neg(\exists x)(\neg\phi)[\alpha]$

for any assignment α

PROOF. We have that:

$\mathcal{M} \vDash (\forall x)\phi[\alpha]$ iff for all $b \in M$ we have that $\mathcal{M} \vDash \phi_1[\alpha_{b/x}]$
iff for all $b \in M$ we have that $\mathcal{M} \nvDash \neg\phi_1[\alpha_{b/x}]$
iff there is no $b \in M$ for which we have $\mathcal{M} \vDash \neg\phi_1[\alpha_{b/x}]$
iff it is not the case that there is $b \in M$ s.t. $\mathcal{M} \vDash \neg\phi_1[\alpha_{b/x}]$
iff it is not the case that $\mathcal{M} \vDash (\exists x)\neg\phi_1[\alpha]$
iff $\mathcal{M} \vDash \neg(\exists x)(\neg\phi)$,

which is what we wanted to show.[13] □

Let's see something we have seen before:

THEOREM 2.2.7. *Let $\phi(x_1, \ldots, x_n)$ be an $\mathcal{L}$-formula and $\mathcal{M}$ an $\mathcal{L}$-structure. For any assignments α, β such that for all $i \leq n$ we have that $\alpha(x_i) = \beta(x_i)$, we have that $\mathcal{M} \vDash \phi[\alpha]$ if and only if $\mathcal{M} \vDash \phi[\beta]$.*

PROOF. The proof is by induction on ϕ. I will only write down the case when ϕ is of the form $(\forall x)\psi$ (see next exercise). Let α and β be as above. Let $c \in M$ be any element. Observe that if α and β agree on all free variables of ϕ, then so do $\alpha_{c/x}$ and $\beta_{c/x}$. Thus:

$$\begin{aligned} \mathcal{M} \vDash (\forall x)\psi[\alpha] &\text{ iff exists } c \in M \text{ s.t. } \mathcal{M} \vDash \psi[\alpha_{b/x}] \\ &\text{ iff exists } c \in M \text{ s.t. } \mathcal{M} \nvDash \neg\psi_1[\beta_{c/x}] \\ &\text{ iff } \mathcal{M} \vDash \neg(\exists x)\psi[\beta]. \end{aligned}$$

□

Exercise 2.2.8. Write out the full proof of the theorem above (you don't have to rewrite the part of the proof that I wrote up, but you should certainly discuss what happens with terms and atomic formulas).

Similarly to propositional sentences, we have the following:

Corollary 2.2.9. *If ϕ is an $\mathcal{L}$-sentence, then for any $\mathcal{L}$-structure $\mathcal{M}$ we have that either:*

(1) For any assignment $\alpha : \mathsf{Var} \to \mathcal{M}$ we have that $\mathcal{M} \vDash \phi[\alpha]$.

(2) For any assignment $\alpha : \mathsf{Var} \to \mathcal{M}$ we have that $\mathcal{M} \nvDash \phi[\alpha]$.

Thus, if ϕ is a sentence, we are justified in writing $\mathcal{M} \vDash \phi$ to indicate that we are in case *(1)* above.

[13]A very minor, but possibly eventually major, point is that to move between the third and fourth equivalences, we used the assumption that our structures are non-empty.