

## Chapter udf

# Introduction to First-Order Logic

### int.1 First-Order Logic

fol:int:fol:  
sec You are probably familiar with first-order logic from your first introduction to formal logic.<sup>1</sup> You may know it as “quantificational logic” or “predicate logic.” First-order logic, first of all, is a formal language. That means, it has a certain vocabulary, and its expressions are strings from this vocabulary. But not every string is permitted. There are different kinds of permitted expressions: terms, **formulas**, and **sentences**. We are mainly interested in **sentences** of first-order logic: they provide us with a formal analogue of sentences of English, and about them we can ask the questions a logician typically is interested in. For instance:

- Does  $\psi$  follow from  $\varphi$  logically?
- Is  $\varphi$  logically true, logically false, or contingent?
- Are  $\varphi$  and  $\psi$  equivalent?

These questions are primarily questions about the “meaning” of **sentences** of first-order logic. For instance, a philosopher would analyze the question of whether  $\psi$  follows logically from  $\varphi$  as asking: is there a case where  $\varphi$  is true but  $\psi$  is false ( $\psi$  doesn’t follow from  $\varphi$ ), or does every case that makes  $\varphi$  true also make  $\psi$  true ( $\psi$  does follow from  $\varphi$ )? But we haven’t been told yet what a “case” is—that is the job of *semantics*. The semantics of first-order logic provides a mathematically precise model of the philosopher’s intuitive idea of “case,” and also—and this is important—of what it is for a **sentence**  $\varphi$  to be *true in* a case. We call the mathematically precise model that we will develop a **structure**. The relation which makes “true in” precise, is called the relation of *satisfaction*. So what we will define is “ $\varphi$  is satisfied in  $\mathfrak{M}$ ” (in symbols:  $\mathfrak{M} \models \varphi$ ) for **sentences**  $\varphi$  and **structures**  $\mathfrak{M}$ . Once this is done, we can also give precise

---

<sup>1</sup>In fact, we more or less assume you are! If you’re not, you could review a more elementary textbook, such as *forall x* (Magnus et al., 2021).

definitions of the other semantical terms such as “follows from” or “is logically true.” These definitions will make it possible to settle, again with mathematical precision, whether, e.g.,  $\forall x (\varphi(x) \rightarrow \psi(x)), \exists x \varphi(x) \models \exists x \psi(x)$ . The answer will, of course, be “yes.” If you’ve already been trained to symbolize sentences of English in first-order logic, you will recognize this as, e.g., the symbolizations of, say, “All ants are insects, there are ants, therefore there are insects.” That is obviously a valid argument, and so our mathematical model of “follows from” for our formal language should give the same answer.

Another topic you probably remember from your first introduction to formal logic is that there are *derivations*. If you have taken a first formal logic course, your instructor will have made you practice finding such *derivations*, perhaps even a *derivation* that shows that the above entailment holds. There are many different ways to give *derivations*: you may have done something called “natural deduction” or “truth trees,” but there are many others. The purpose of *derivation* systems is to provide tools using which the logicians’ questions above can be answered: e.g., a natural deduction *derivation* in which  $\forall x (\varphi(x) \rightarrow \psi(x))$  and  $\exists x \varphi(x)$  are premises and  $\exists x \psi(x)$  is the conclusion (last line) *verifies* that  $\exists x \psi(x)$  logically follows from  $\forall x (\varphi(x) \rightarrow \psi(x))$  and  $\exists x \varphi(x)$ .

But why is that? On the face of it, *derivation* systems have nothing to do with semantics: giving a formal *derivation* merely involves arranging symbols in certain rule-governed ways; they don’t mention “cases” or “true in” at all. The connection between *derivation* systems and semantics has to be established by a meta-logical investigation. What’s needed is a mathematical proof, e.g., that a formal *derivation* of  $\exists x \psi(x)$  from premises  $\forall x (\varphi(x) \rightarrow \psi(x))$  and  $\exists x \varphi(x)$  is possible, if, and only if,  $\forall x (\varphi(x) \rightarrow \psi(x))$  and  $\exists x \varphi(x)$  together entails  $\exists x \psi(x)$ . Before this can be done, however, a lot of painstaking work has to be carried out to get the definitions of syntax and semantics correct.

## int.2 Syntax

We first must make precise what strings of symbols count as *sentences* of first-order logic. We’ll do this later; for now we’ll just proceed by example. The basic building blocks—the vocabulary—of first-order logic divides into two parts. The first part is the symbols we use to say specific things or to pick out specific things. We pick out things using *constant symbols*, and we say stuff about the things we pick out using *predicate symbols*. E.g, we might use  $a$  as a *constant symbol* to pick out a single thing, and then say something about it using the *sentence*  $P(a)$ . If you have meanings for “ $a$ ” and “ $P$ ” in mind, you can read  $P(a)$  as a sentence of English (and you probably have done so when you first learned formal logic). Once you have such simple *sentences* of first-order logic, you can build more complex ones using the second part of the vocabulary: the logical symbols (connectives and quantifiers). So, for instance, we can form expressions like  $(P(a) \wedge Q(b))$  or  $\exists x P(x)$ .

In order to provide the precise definitions of semantics and the rules of our *derivation* systems required for rigorous meta-logical study, we first of all

fol:int:syn:  
sec

have to give a precise definition of what counts as a **sentence** of first-order logic. The basic idea is easy enough to understand: there are some simple **sentences** we can form from just **predicate symbols** and **constant symbols**, such as  $P(a)$ . And then from these we form more complex ones using the connectives and quantifiers. But what exactly are the rules by which we are allowed to form more complex **sentences**? These must be specified, otherwise we have not defined “**sentence** of first-order logic” precisely enough. There are a few issues. The first one is to get the right strings to count as **sentences**. The second one is to do this in such a way that we can give mathematical proofs about *all* **sentences**. Finally, we’ll have to also give precise definitions of some rudimentary operations with **sentences**, such as “replace every  $x$  in  $\varphi$  by  $b$ .” The trouble is that the quantifiers and **variables** we have in first-order logic make it not entirely obvious how this should be done. E.g., should  $\exists x P(a)$  count as a **sentence**? What about  $\exists x \exists x P(x)$ ? What should the result of “replace  $x$  by  $b$  in  $(P(x) \wedge \exists x P(x))$ ” be?

### int.3 Formulas

Here is the approach we will use to rigorously specify **sentences** of first-order logic and to deal with the issues arising from the use of **variables**. We first define a *different* set of expressions: **formulas**. Once we’ve done that, we can consider the role **variables** play in them—and on the basis of some other ideas, namely those of “free” and “bound” **variables**, we can define what a **sentence** is (namely, a **formula** without free **variables**). We do this not just because it makes the definition of “**sentence**” more manageable, but also because it will be crucial to the way we define the semantic notion of satisfaction.

Let’s define “**formula**” for a simple first-order language, one containing only a single **predicate symbol**  $P$  and a single **constant symbol**  $a$ , and only the logical symbols  $\neg$ ,  $\wedge$ , and  $\exists$ . Our full definitions will be much more general: we’ll allow infinitely many **predicate symbols** and **constant symbols**. In fact, we will also consider **function symbols** which can be combined with **constant symbols** and **variables** to form “terms.” For now,  $a$  and the variables will be our only terms. We do need infinitely many **variables**. We’ll officially use the symbols  $v_0, v_1, \dots$ , as variables.

**Definition int.1.** The set of *formulas*  $\text{Frm}$  is defined as follows:

fol:int:fml:  
fmls-atom

1.  $P(a)$  and  $P(v_i)$  are **formulas** ( $i \in \mathbb{N}$ ).
2. If  $\varphi$  is a **formula**, then  $\neg\varphi$  is **formula**.
3. If  $\varphi$  and  $\psi$  are **formulas**, then  $(\varphi \wedge \psi)$  is a **formula**.
4. If  $\varphi$  is a **formula** and  $x$  is a **variable**, then  $\exists x \varphi$  is a **formula**.

fol:int:fml:  
fmls-limit

5. Nothing else is a **formula**.

(1) tell us that  $P(a)$  and  $P(v_i)$  are **formulas**, for any  $i \in \mathbb{N}$ . These are the so-called *atomic formulas*. They give us something to start from. The other clauses give us ways of forming new **formulas** from ones we have already formed. So for instance, we get that  $\neg P(v_2)$  is a **formula**, since  $P(v_2)$  is already a **formula** by (1), and then we get that  $\exists v_2 \neg P(v_2)$  is another **formula**, and so on. (5) tells us that *only* strings we can form in this way count as **formulas**. In particular,  $\exists v_0 P(a)$  and  $\exists v_0 \exists v_0 P(a)$  *do* count as **formulas**, and  $(\neg P(a))$  does not.

This way of defining **formulas** is called an *inductive definition*, and it allows us to prove things about **formulas** using a version of proof by induction called *structural induction*. These are discussed in a general way in ?? and ??, which you should review before delving into the proofs later on. Basically, the idea is that if you want to give a proof that something is true for all **formulas** you show first that it is true for the atomic **formulas**, and then that *if* it's true for any **formula**  $\varphi$  (and  $\psi$ ), it's *also* true for  $\neg\varphi$ ,  $(\varphi \wedge \psi)$ , and  $\exists x \varphi$ . For instance, this proves that it's true for  $\exists v_2 \neg P(v_2)$ : from the first part you know that it's true for the atomic **formula**  $P(v_2)$ . Then you get that it's true for  $\neg P(v_2)$  by the second part, and then again that it's true for  $\exists v_2 \neg P(v_2)$  itself. Since all **formulas** are inductively generated from atomic **formulas**, this works for any of them.

## int.4 Satisfaction

We can already skip ahead to the semantics of first-order logic once we know what **formulas** are: here, the basic definition is that of a **structure**. For our simple language, a **structure**  $\mathfrak{M}$  has just three components: a non-empty set  $|\mathfrak{M}|$  called the *domain*, what  $a$  picks out in  $\mathfrak{M}$ , and what  $P$  is true of in  $\mathfrak{M}$ . The object picked out by  $a$  is denoted  $a^{\mathfrak{M}}$  and the set of things  $P$  is true of by  $P^{\mathfrak{M}}$ . A **structure**  $\mathfrak{M}$  consists of just these three things:  $|\mathfrak{M}|$ ,  $a^{\mathfrak{M}} \in |\mathfrak{M}|$  and  $P^{\mathfrak{M}} \subseteq |\mathfrak{M}|$ . The general case will be more complicated, since there will be many **predicate symbols** and **constant symbols**, the **constant symbols** can have more than one place, and there will also be **function symbols**.

fol:int:sat:  
sec

This is enough to give a definition of satisfaction for **formulas** that don't contain **variables**. The idea is to give an inductive definition that mirrors the way we have defined **formulas**. We specify when an atomic formula is satisfied in  $\mathfrak{M}$ , and then when, e.g.,  $\neg\varphi$  is satisfied in  $\mathfrak{M}$  on the basis of whether or not  $\varphi$  is satisfied in  $\mathfrak{M}$ . E.g., we could define:

1.  $P(a)$  is satisfied in  $\mathfrak{M}$  iff  $a^{\mathfrak{M}} \in P^{\mathfrak{M}}$ .
2.  $\neg\varphi$  is satisfied in  $\mathfrak{M}$  iff  $\varphi$  is not satisfied in  $\mathfrak{M}$ .
3.  $(\varphi \wedge \psi)$  is satisfied in  $\mathfrak{M}$  iff  $\varphi$  is satisfied in  $\mathfrak{M}$ , and  $\psi$  is satisfied in  $\mathfrak{M}$  as well.

Let's say that  $|\mathfrak{M}| = \{0, 1, 2\}$ ,  $a^{\mathfrak{M}} = 1$ , and  $P^{\mathfrak{M}} = \{1, 2\}$ . This definition would tell us that  $P(a)$  is satisfied in  $\mathfrak{M}$  (since  $a^{\mathfrak{M}} = 1 \in \{1, 2\} = P^{\mathfrak{M}}$ ). It tells us

further that  $\neg P(a)$  is not satisfied in  $\mathfrak{M}$ , and that in turn that  $\neg\neg P(a)$  is and  $(\neg P(a) \wedge P(a))$  is not satisfied, and so on.

The trouble comes when we want to give a definition for the quantifiers: we'd like to say something like, “ $\exists v_0 P(v_0)$  is satisfied iff  $P(v_0)$  is satisfied.” But the **structure**  $\mathfrak{M}$  doesn't tell us what to do about **variables**. What we actually want to say is that  $P(v_0)$  is satisfied *for some value of*  $v_0$ . To make this precise we need a way to assign **elements** of  $|\mathfrak{M}|$  not just to  $a$  but also to  $v_0$ . To this end, we introduce **variable assignments**. A **variable assignment** is simply a function  $s$  that maps **variables** to **elements** of  $|\mathfrak{M}|$  (in our example, to one of 1, 2, or 3). Since we don't know beforehand which **variables** might appear in a **formula** we can't limit which **variables**  $s$  assigns values to. The simple solution is to require that  $s$  assigns values to *all* **variables**  $v_0, v_1, \dots$ . We'll just use only the ones we need.

Instead of defining satisfaction of **formulas** just relative to a **structure**, we'll define it relative to a **structure**  $\mathfrak{M}$  *and* a **variable assignment**  $s$ , and write  $\mathfrak{M}, s \models \varphi$  for short. Our definition will now include an additional clause to deal with atomic **formulas** containing **variables**:

1.  $\mathfrak{M}, s \models P(a)$  iff  $a^{\mathfrak{M}} \in P^{\mathfrak{M}}$ .
2.  $\mathfrak{M}, s \models P(v_i)$  iff  $s(v_i) \in P^{\mathfrak{M}}$ .
3.  $\mathfrak{M}, s \models \neg\varphi$  iff not  $\mathfrak{M}, s \models \varphi$ .
4.  $\mathfrak{M}, s \models (\varphi \wedge \psi)$  iff  $\mathfrak{M}, s \models \varphi$  and  $\mathfrak{M}, s \models \psi$ .

Ok, this solves one problem: we can now say when  $\mathfrak{M}$  satisfies  $P(v_0)$  for the value  $s(v_0)$ . To get the definition right for  $\exists v_0 P(v_0)$  we have to do one more thing: We want to have that  $\mathfrak{M}, s \models \exists v_0 P(v_0)$  iff  $\mathfrak{M}, s' \models P(v_0)$  for *some* way  $s'$  of assigning a value to  $v_0$ . But the value assigned to  $v_0$  does not necessarily have to be the value that  $s(v_0)$  picks out. We'll introduce a notation for that: if  $m \in |\mathfrak{M}|$ , then we let  $s[m/v_0]$  be the assignment that is just like  $s$  (for all **variables** other than  $v_0$ ), except to  $v_0$  it assigns  $m$ . Now our definition can be:

5.  $\mathfrak{M}, s \models \exists v_i \varphi$  iff  $\mathfrak{M}, s[m/v_i] \models \varphi$  for some  $m \in |\mathfrak{M}|$ .

Does it work out? Let's say we let  $s(v_i) = 0$  for all  $i \in \mathbb{N}$ .  $\mathfrak{M}, s \models \exists v_0 P(v_0)$  iff there is an  $m \in |\mathfrak{M}|$  so that  $\mathfrak{M}, s[m/v_0] \models P(v_0)$ . And there is: we can choose  $m = 1$  or  $m = 2$ . Note that this is true even if the value  $s(v_0)$  assigned to  $v_0$  by  $s$  itself—in this case, 0—doesn't do the job. We have  $\mathfrak{M}, s[1/v_0] \models P(v_0)$  but not  $\mathfrak{M}, s \models P(v_0)$ .

If this looks confusing and cumbersome: it is. But the added complexity is required to give a precise, inductive definition of satisfaction for all **formulas**, and we need something like it to precisely define the semantic notions. There are other ways of doing it, but they are all equally (in)legant.

## int.5 Sentences

Ok, now we have a (sketch of a) definition of satisfaction (“true in”) for **structures** and **formulas**. But it needs this additional bit—a **variable** assignment—and what we wanted is a definition of **sentences**. How do we get rid of assignments, and what are **sentences**?

fol:int:snt:  
sec

You probably remember a discussion in your first introduction to formal logic about the relation between **variables** and quantifiers. A quantifier is always followed by a **variable**, and then in the part of the **sentence** to which that quantifier applies (its “scope”), we understand that the **variable** is “bound” by that quantifier. In **formulas** it was not required that every **variable** has a matching quantifier, and **variables** without matching quantifiers are “free” or “unbound.” We will take **sentences** to be all those **formulas** that have no free **variables**.

Again, the intuitive idea of when an occurrence of a **variable** in a **formula**  $\varphi$  is bound, which quantifier binds it, and when it is free, is not difficult to get. You may have learned a method for testing this, perhaps involving counting parentheses. We have to insist on a precise definition—and because we have defined **formulas** by induction, we can give a definition of the free and bound occurrences of a **variable**  $x$  in a **formula**  $\varphi$  also by induction. E.g., it might look like this for our simplified language:

1. If  $\varphi$  is atomic, all occurrences of  $x$  in it are free (that is, the occurrence of  $x$  in  $P(x)$  is free).
2. If  $\varphi$  is of the form  $\neg\psi$ , then an occurrence of  $x$  in  $\neg\psi$  is free iff the corresponding occurrence of  $x$  is free in  $\psi$  (that is, the free occurrences of variables in  $\psi$  are exactly the corresponding occurrences in  $\neg\psi$ ).
3. If  $\varphi$  is of the form  $(\psi \wedge \chi)$ , then an occurrence of  $x$  in  $(\psi \wedge \chi)$  is free iff the corresponding occurrence of  $x$  is free in  $\psi$  or in  $\chi$ .
4. If  $\varphi$  is of the form  $\exists x \psi$ , then no occurrence of  $x$  in  $\varphi$  is free; if it is of the form  $\exists y \psi$  where  $y$  is a different **variable** than  $x$ , then an occurrence of  $x$  in  $\exists y \psi$  is free iff the corresponding occurrence of  $x$  is free in  $\psi$ .

Once we have a precise definition of free and bound occurrences of variables, we can simply say: a **sentence** is any **formula** without free occurrences of **variables**.

## int.6 Semantic Notions

We mentioned above that when we consider whether  $\mathfrak{M}, s \models \varphi$  holds, we (for convenience) let  $s$  assign values to all **variables**, but only the values it assigns to **variables** in  $\varphi$  are used. In fact, it’s only the values of *free* variables in  $\varphi$  that matter. Of course, because we’re careful, we are going to prove this fact. Since **sentences** have no free variables,  $s$  doesn’t matter at all when it comes to

fol:int:sem:  
sec

whether or not they are satisfied in a **structure**. So, when  $\varphi$  is a **sentence** we can define  $\mathfrak{M} \models \varphi$  to mean “ $\mathfrak{M}, s \models \varphi$  for all  $s$ ,” which as it happens is true iff  $\mathfrak{M}, s \models \varphi$  for at least one  $s$ . We need to introduce **variable** assignments to get a working definition of satisfaction for **formulas**, but for **sentences**, satisfaction is independent of the **variable** assignments.

Once we have a definition of “ $\mathfrak{M} \models \varphi$ ,” we know what “case” and “true in” mean as far as **sentences** of first-order logic are concerned. On the basis of the definition of  $\mathfrak{M} \models \varphi$  for **sentences** we can then define the basic semantic notions of validity, entailment, and satisfiability. A sentence is valid,  $\models \varphi$ , if every **structure** satisfies it. It is entailed by a set of **sentences**,  $\Gamma \models \varphi$ , if every **structure** that satisfies all the **sentences** in  $\Gamma$  also satisfies  $\varphi$ . And a set of **sentences** is satisfiable if some **structure** satisfies all **sentences** in it at the same time.

Because **formulas** are inductively defined, and satisfaction is in turn defined by induction on the structure of **formulas**, we can use induction to prove properties of our semantics and to relate the semantic notions defined. We’ll collect and prove some of these properties, partly because they are individually interesting, but mainly because many of them will come in handy when we go on to investigate the relation between semantics and **derivation** systems. In order to do so, we’ll also have to define (precisely, i.e., by induction) some syntactic notions and operations we haven’t mentioned yet.

## int.7 Substitution

fol:int:sub:  
sec

We’ll discuss an example to illustrate how things hang together, and how the development of syntax and semantics lays the foundation for our more advanced investigations later. Our **derivation** systems should let us **derive**  $P(a)$  from  $\forall v_0 P(v_0)$ . Maybe we even want to state this as a rule of inference. However, to do so, we must be able to state it in the most general terms: not just for  $P$ ,  $a$ , and  $v_0$ , but for any **formula**  $\varphi$ , and term  $t$ , and **variable**  $x$ . (Recall that **constant symbols** are terms, but we’ll consider also more complicated terms built from **constant symbols** and **function symbols**.) So we want to be able to say something like, “whenever you have **derived**  $\forall x \varphi(x)$  you are justified in inferring  $\varphi(t)$ —the result of removing  $\forall x$  and replacing  $x$  by  $t$ .” But what exactly does “replacing  $x$  by  $t$ ” mean? What is the relation between  $\varphi(x)$  and  $\varphi(t)$ ? Does this always work?

To make this precise, we define the operation of *substitution*. Substitution is actually tricky, because we can’t just replace all  $x$ ’s in  $\varphi$  by  $t$ , and not every  $t$  can be substituted for any  $x$ . We’ll deal with this, again, using inductive definitions. But once this is done, specifying an inference rule as “infer  $\varphi(t)$  from  $\forall x \varphi(x)$ ” becomes a precise definition. Moreover, we’ll be able to show that this is a good inference rule in the sense that  $\forall x \varphi(x)$  entails  $\varphi(t)$ . But to prove this, we have to again prove something that may at first glance prompt you to ask “why are we doing this?” That  $\forall x \varphi(x)$  entails  $\varphi(t)$  relies on the fact that whether or not  $\mathfrak{M} \models \varphi(t)$  holds depends only on the value of the term  $t$ ,

i.e., if we let  $m$  be whatever **element** of  $|\mathfrak{M}|$  is picked out by  $t$ , then  $\mathfrak{M}, s \models \varphi(t)$  iff  $\mathfrak{M}, s[m/x] \models \varphi(x)$ . This holds even when  $t$  contains **variables**, but we'll have to be careful with how exactly we state the result.

## int.8 Models and Theories

Once we've defined the syntax and semantics of first-order logic, we can get to work investigating the properties of **structures**, of the semantic notions, we can define **derivation** systems, and investigate those. For a set of **sentences**, we can ask: what **structures** make all the **sentences** in that set true? Given a set of **sentences**  $\Gamma$ , a **structure**  $\mathfrak{M}$  that satisfies them is called a *model of  $\Gamma$* . We might start from  $\Gamma$  and try find its models—what do they look like? How big or small do they have to be? But we might also start with a single **structure** or collection of **structures** and ask: what **sentences** are true in them? Are there **sentences** that *characterize* these **structures** in the sense that they, and only they, are true in them? These kinds of questions are the domain of *model theory*. They also underlie the *axiomatic method*: describing a collection of **structures** by a set of **sentences**, the axioms of a theory. This is made possible by the observation that exactly those **sentences** entailed in first-order logic by the axioms are true in all models of the axioms. fol:int:mod:  
sec

As a very simple example, consider preorders. A preorder is a relation  $R$  on some set  $A$  which is both reflexive and transitive. A set  $A$  with a two-place relation  $R \subseteq A \times A$  on it is exactly what we would need to give a **structure** for a first-order language with a single two-place relation symbol  $P$ : we would set  $|\mathfrak{M}| = A$  and  $P^{\mathfrak{M}} = R$ . Since  $R$  is a preorder, it is reflexive and transitive, and we can find a set  $\Gamma$  of **sentences** of first-order logic that say this:

$$\begin{aligned} &\forall v_0 P(v_0, v_0) \\ &\forall v_0 \forall v_1 \forall v_2 ((P(v_0, v_1) \wedge P(v_1, v_2)) \rightarrow P(v_0, v_2)) \end{aligned}$$

These **sentences** are just the symbolizations of “for any  $x$ ,  $Rxx$ ” ( $R$  is reflexive) and “whenever  $Rxy$  and  $Ryz$  then also  $Rxz$ ” ( $R$  is transitive). We see that a **structure**  $\mathfrak{M}$  is a model of these two **sentences**  $\Gamma$  iff  $R$  (i.e.,  $P^{\mathfrak{M}}$ ), is a preorder on  $A$  (i.e.,  $|\mathfrak{M}|$ ). In other words, the models of  $\Gamma$  are exactly the preorders. Any property of all preorders that can be expressed in the first-order language with just  $P$  as **predicate symbol** (like reflexivity and transitivity above), is entailed by the two **sentences** in  $\Gamma$  and vice versa. So anything we can prove about models of  $\Gamma$  we have proved about all preorders.

For any particular theory and class of models (such as  $\Gamma$  and all preorders), there will be interesting questions about what can be expressed in the corresponding first-order language, and what cannot be expressed. There are some properties of **structures** that are interesting for all languages and classes of models, namely those concerning the size of the **domain**. One can always express, for instance, that the **domain** contains exactly  $n$  **elements**, for any  $n \in \mathbb{Z}^+$ . One can also express, using a set of infinitely many **sentences**, that the **domain** is infinite. But one cannot express that the domain is finite, or that the domain



is **non-enumerable**. These results about the limitations of first-order languages are consequences of the compactness and Löwenheim-Skolem theorems.

## int.9 Soundness and Completeness

fol:int:scp:  
sec

We'll also introduce **derivation** systems for first-order logic. There are many **derivation** systems that logicians have developed, but they all define the same **derivability** relation between **sentences**. We say that  $\Gamma$  *derives*  $\varphi$ ,  $\Gamma \vdash \varphi$ , if there is a **derivation** of a certain precisely defined sort. **Derivations** are always finite arrangements of symbols—perhaps a list of **sentences**, or some more complicated structure. The purpose of **derivation** systems is to provide a tool to determine if a **sentence** is entailed by some set  $\Gamma$ . In order to serve that purpose, it must be true that  $\Gamma \models \varphi$  if, and only if,  $\Gamma \vdash \varphi$ .

If  $\Gamma \vdash \varphi$  but not  $\Gamma \models \varphi$ , our **derivation** system would be too strong, prove too much. The property that if  $\Gamma \vdash \varphi$  then  $\Gamma \models \varphi$  is called *soundness*, and it is a minimal requirement on any good **derivation** system. On the other hand, if  $\Gamma \models \varphi$  but not  $\Gamma \vdash \varphi$ , then our **derivation** system is too weak, it doesn't prove enough. The property that if  $\Gamma \models \varphi$  then  $\Gamma \vdash \varphi$  is called *completeness*. Soundness is usually relatively easy to prove (by induction on the structure of **derivations**, which are inductively defined). Completeness is harder to prove.

Soundness and completeness have a number of important consequences. If a set of **sentences**  $\Gamma$  *derives* a contradiction (such as  $\varphi \wedge \neg\varphi$ ) it is called *inconsistent*. Inconsistent  $\Gamma$ s cannot have any models, they are unsatisfiable. From completeness the converse follows: any  $\Gamma$  that is not inconsistent—or, as we will say, *consistent*—has a model. In fact, this is equivalent to completeness, and is the form of completeness we will actually prove. It is a deep and perhaps surprising result: just because you cannot prove  $\varphi \wedge \neg\varphi$  from  $\Gamma$  guarantees that there is a **structure** that is as  $\Gamma$  describes it. So completeness gives an answer to the question: which sets of **sentences** have models? Answer: all and only consistent sets do.

The soundness and completeness theorems have two important consequences: the compactness and the Löwenheim-Skolem theorem. These are important results in the theory of models, and can be used to establish many interesting results. We've already mentioned two: first-order logic cannot express that the **domain** of a **structure** is finite or that it is **non-enumerable**.

Historically, all of this—how to define syntax and semantics of first-order logic, how to define good **derivation** systems, how to prove that they are sound and complete, getting clear about what can and cannot be expressed in first-order languages—took a long time to figure out and get right. We now know how to do it, but going through all the details can still be confusing and tedious. But it's also important, because the methods developed here for the formal language of first-order logic are applied all over the place in logic, computer science, and linguistics. So working through the details pays off in the long

run.

## Photo Credits

# Bibliography

Magnus, P. D., Tim Button, J. Robert Loftis, Aaron Thomas-Bolduc, Robert Trueman, and Richard Zach. 2021. *forall x: Calgary. An Introduction to Formal Logic*. Calgary: Open Logic Project, f21 ed. URL <https://forallx.openlogicproject.org/>.