3.1: First Order Logic Syntax and Semantics

Observe first that logic is not the study of truth, but of the relationship between the truth of one statement and that of another. That is, in logic, we do not care whether a statement like “If angels exist then necessarily all of them fit on the head of a needle” (suitably formalized) is indeed true in reality, but if the if-part were true (resp., false), then what does that say about the truth value of the then-part of the statement? And likewise for a whole bunch of such sentences. Others do care what is represented formally with such a logic language, but we will defer that to Block II.

To be able to study those aspects of logic, we need a language that is unambiguous; natural language is not. You may have encountered propositional logic already, and first order predicate logic (FOL) is an extension of that, which enables us to represent more knowledge in more detail. Here, I will give only a brief glimpse of it. Eventually, you will need to be able to recognize, understand, and be able to formalize at least a little bit in FOL. Of all the definitions that will follow shortly, there are four important ideas to grasp: the syntax of a language, the model-theoretic semantics of a language, what a theory means in the context of logic, and the notion of deduction where we apply some rules to what has been represented explicitly so as to derive knowledge that was represented only implicitly. We will address each in turn.

First, there are two principal components to consider for the language:

Definition \(\PageIndex{1}\): Syntax

Syntax has to do with what ‘things’ (symbols, notations) one is allowed to use in the language and in what way; there is/are a(n):

- Alphabet
- Language constructs
- Sentences to assert knowledge
Definition \((\PageIndex{2})\): Semantics

Semantics: Formal meaning, which has to do what those sentences with the alphabet and constructs are supposed to mean.

Their details are presented in the remainder of this section.

**Syntax**

The lexicon of a first order language contains the following:

- Connectives and Parentheses: \(\neg\), \(\to\), \(\leftarrow\), \(\wedge\), \(\vee\), ( and );
- Quantifiers: \(\forall\) (universal) and \(\exists\) (existential);
- Variables: \(x, y, z, ...\) ranging over particulars (individual objects);
- Constants: \(a, b, c, ...\) representing a specific element;
- Functions: \(f, g, h, ...\), with arguments listed as \(f(x_{1}, ...x _{n})\);

There is an (countably infinite) supply of **symbols** (signature): variables, functions, constants, and relations.

In other words: we can use these things to create 'sentences', like we have in natural language, but then controlled and with a few extra figurines. Let us look first at how we can formalise a natural language sentence into first order logic.

Example \((\PageIndex{1})\):

From Natural Language to First order logic (or vv.). Consider the following three sentences:

- “Each animal is an organism”
- “All animals are organisms”
- “If it is an animal then it is an organism”

This can be formalised as:

\[
\forall x (\text{Animal}(x) \to \text{Organism}(x))\]

Observe the colour coding in the natural language sentences: ‘each’ and ‘all’ are different ways to say “\(\forall\)” and the ‘is a’ and ‘are’ in the first two sentences match the “\(\to\)”, and the combination of the “\(\forall\)” and “\(\to\)” in this particular construction can be put into natural language as ‘if ... then’.

Instead of talking about all objects of a particular type, one also can assert there are at least some of them; e.g.,

- “Aliens exist”

could be formalised as
with the ‘exist’ matching the ∃, and

- “There are books that are heavy”

(Well, at least one of them is) as:

∃x(Book(x) ∧ heavy(x))

where the ‘there are’ is another way to talk about "(∃x)" and the ‘that’ hides (linguistically) the "(∧)". A formulation like “There is at least one book, and it is heavy” is a natural language rendering that is closer to the structure of the axiom.

A sentence—or, more precisely, its precise meaning—such as "Each student must be registered for a degree programme" requires a bit more consideration. There are at least two ways to say the same thing in the way the sentence is formulated (we leave the arguments for and against each option for another time):

1. ∀x, y(registered for(x, y) → Student(x) ∧ DegreeProgramme(y))

   “if there is a registered for relation, then the first object is a student and the second one a degree programme”

∀x(Student(x) → ∃y registered for(x, y))

   “Each student is registered for at least one y”, where the y is a degree programme (taken from the first axiom)

2. ∀x(Student(x) → ∃y (registered for(x, y) ∧ DegreeProgramme(y))))

   “Each student is registered for at least one degree programme’

But all this is still just syntax (it does not say what it really means), and it looks like a ‘free for all’ on how we can use these symbols. This is, in fact, not the case, and the remainder of the definitions will make this more precise, which will be illustrated in Example 2.1.2 afterward.

There is a systematic to the axioms in the examples, with the brackets, arrows, etc. Let us put this more precisely now. We start from the basic elements and gradually build it up to more complex things.

Definition ∃x

(Term) A term is inductively defined by two rules, being:

- Every variable and constant is a term.
- if f is a m-ary function and t1, ..., tm are terms, then f(t1, ..., tm) is also a term.

Definition ∀x

(Atomic formula) An atomic formula is a formula that has the form \(t1 = t2\) or \(R(t_{1}, ..., t_{n})\) where R is an n-ary relation and \(t_{1}, ..., t_{n}\) are terms.
Definition \(\PageIndex{5}\)

(Formula) A string of symbols is a formula of FOL if and only if it is constructed from atomic formulas by repeated applications of rules R1, R2, and R3.

R1. If \(\phi\) is a formula then so is \(\neg\phi\).

R2. If \(\phi\) and \(\psi\) are formulas then so is \(\phi\wedge\psi\).

R3. If \(\phi\) is a formula then so is \(\exists x \phi\) for any variable \(x\).

A free variable of a formula \(\phi\) is that variable occurring in \(\phi\) that is not quantified. For instance, if \(\phi = \forall x (\text{Loves}(x,y))\), then \(y\) is the free variable, as it is not bound to a quantifier. We now can introduce the definition of sentence.

Definition \(\PageIndex{6}\)

(Sentence) A sentence of FOL is a formula having no free variables.

Check that there are no free variables in the axioms in Example 2.1.1, i.e., they are all sentences.

Up to this point, we have seen only a few examples with going back-and-forth between sentences in natural language and in FOL. This is by no means to only option. One can also formalise diagrams and provide logic-based reconstructions of, say, UML class diagrams in order to be precise. We can already do this with syntax introduced so far.

Let’s consider again the lion eating impalas and herbivores, as was shown in Figure 1.1.1-B in Chapter 1. First, we ‘bump up’ the eats association end to the name of the binary relation, eats. Second, noting that UML uses lookacross notation for its associations, the \(1..*\) ’at least one’ amounts to an existential quantification for the impalas, and the * ‘zero or more’ to a universal quantification for the herbivores. This brings us back to Eq. 1.1.1 from Chapter 1:

\[
\forall x (\text{Lion}(x) \rightarrow \forall y (\text{eats}(x,y) \land \text{Herbivore}(y)) \land \exists z (\text{eats}(x,z) \land \text{Impala}))\]

Another example with different constraints is shown in Figure 2.1.1, which could be the start of a UML class diagram (but incomplete; notably because it has no attributes and no methods) or an abuse of notation, in that an ontology is shown diagrammatically in UML class diagram notation, as there is no official graphical language to depict ontologies.

\[\text{Figure 2.1.1: A UML model that can be formally represented in FOL; see text for details.}\]

Either way, syntactically, also here the UML classes can be converted into unary predicates in FOL, the association is translated into a binary relation, and the multiplicity constraint turns into an existential quantification that is restricted to exactly 4 limbs\(^2\). In the interest of succinctness and convention, the latter is permitted to be abbreviated as

\[\text{https://eng.libretexts.org/Bookshelves/Computer_{Science}/Book\%3A_{An}_{Introduction}_{to}_{Ontology}_{Engineering}_{(Keet)/03\%3…}\]

Updated: Tue, 09 Jun 2020 14:21:45 GMT
Powered by
4
The corresponding sentence in FOL is listed in Eq. 2.1.5. One can dwell on the composite aggregation (the black diamond), and we will do so in a later chapter. Here, it is left as an association with an ‘at most one’ constraint on the whole side (Eq. 2.1.6), or: “if there’s a limb related to two animal instances through whole, then those two instances must be the same object”.

\[\forall x (Animal(x) \rightarrow \exists^{=4} y (part(x, y) \land Limb(y)))\]

\[\forall x, y, z (Limb(x) \land whole(x, y) \land whole(x, z) \land Animal(y) \land Animal(z) \rightarrow y = z)\]

That is, indeed, literally talking of two references to one object.

The ‘new’ constraints that we have not translated before yet, are the subclassing, the disjointness, and the completeness. Subclassing is the same as in Eq. 2.1.1, hence, we obtain Eqs. 2.1.7-2.1.9.

\[\forall x (Omnivore(x) \rightarrow Animal(x))\]

\[\forall x (Herbivore(x) \rightarrow Animal(x))\]

\[\forall x (Carnivore(x) \rightarrow Animal(x))\]

Disjoint means that the intersection is empty, so it has the pattern \[\forall x (A(x) \land B(x) \rightarrow \bot)\], where \("\bot\) is the bottom concept/unary predicate that is always false; hence, Eq. 2.1.10 for the sample classes of Figure 2.1.1.

Completeness over the specialisation means that all the instances of the superclass must be an instance of either of the subclasses; hence Eq. 2.1.11.

\[\forall x (Omnivore(x) \land Herbivore(x) \land Carnivore(x) \rightarrow \bot)\]

\[\forall x (Animal(x) \rightarrow Omnivore(x) \lor Herbivore(x) \lor Carnivore(x))\]

These are all syntactic transformations, both by example and that informally some translation rules and a general pattern (like for disjointness) have been noted. From a software engineering viewpoint, this can be seen as a step toward a logic-based reconstruction of UML class diagrams. From a logic viewpoint, the diagram can be seen as ‘syntactic sugar’ for the axioms, which is more accessible to domain experts (non-logicians) than logic.

This playing with syntax, however, does not say what it all means. How do these sentences in FOL map to the objects in, say, the Java application (if it were a UML diagram intended as such) or to some domain of objects wherever that is represented somehow (if it were a UML diagram depicting an ontology)? And can the latter be specified a bit more precise? We shall see the theoretical answers to such question in the next section.

Semantics

Whether a sentence is true or not depends on the underlying set and the interpretation of the function, constant, and relation symbols. To this end, we have structures: a structure consists of an underlying set together with an interpretation of functions, constants, and relations. Given a sentence \(\phi\) and a structure \(\langle M\rangle\), \(\langle M\rangle\) models \(\langle\phi\rangle\).
means that the sentence \(\phi\) is true with respect to \(M\). More precisely, through the following set of definitions (which will be illustrated afterward):

Definition \(\PageIndex{7}\)

(Vocabulary) A vocabulary \(\mathcal{V}\) is a set of function, relation, and constant symbols.

Definition \(\PageIndex{8}\)

(\(\mathcal{V}\)-structure) A \(\mathcal{V}\)-structure consists of a non-empty underlying set \(\Delta\) along with an interpretation of \(\mathcal{V}\). An interpretation of \(\mathcal{V}\) assigns an element of \(\Delta\) to each constant in \(\mathcal{V}\), a function from \(\Delta^n\) to \(\Delta\) to each n-ary function in \(\mathcal{V}\), and a subset of \(\Delta^n\) to each n-ary relation in \(\mathcal{V}\). We say \(M\) is a structure if it is a \(\mathcal{V}\)-structure of some vocabulary \(\mathcal{V}\).

Definition \(\PageIndex{9}\)

(\(\mathcal{V}\)-formula) Let \(\mathcal{V}\) be a vocabulary. A \(\mathcal{V}\)-formula is a formula in which every function, relation, and constant is in \(\mathcal{V}\). A \(\mathcal{V}\)-sentence is a \(\mathcal{V}\)-formula that is a sentence.

When we say that \(M\) models \(\phi\), denoted with \(M \models \phi\), this is with respect to \(M\) being a \(\mathcal{V}\)-structure and \(\mathcal{V}\)-sentence \(\phi\) is true in \(M\).

Model theory is about the interplay between \(M\) and a set of first-order sentences \(\mathcal{T}\) \((M)\), which is called the theory of \(M\), and its ‘inverse’ from a set of sentences \(\Gamma\) to a class of structures.

Definition \(\PageIndex{10}\)

(Theory of \(M\)) For any \(\mathcal{V}\)-structure \(M\), the theory of \(M\), denoted with \(\mathcal{T}(M)\), is the set of all \(\mathcal{V}\)-sentences \(\phi\) such that \(M \models \phi\).

Definition \(\PageIndex{11}\)

(Model) For any set of \(\mathcal{V}\)-sentences, a model of \(\Gamma\) is a \(\mathcal{V}\)-structure that models each sentence in \(\Gamma\). The class of all models of \(\Gamma\) is denoted by \(M(\Gamma)\).

Now we can go to the interesting notions: theory in the context of logic:

Definition \(\PageIndex{12}\)

(Complete \(\mathcal{V}\)-theory) Let \(\mathcal{Gamma}\) be a set of \(\mathcal{V}\)-sentences. Then \(\mathcal{Gamma}\) is a complete \(\mathcal{V}\)-theory if, for any \(\mathcal{V}\)-sentence \(\phi\) either \(\mathcal{Gamma}\) or \(\neg\mathcal{Gamma}\) is in \(\mathcal{Gamma}\) and it is not the case that both \(\phi\) and \(\neg\phi\) are in \(\mathcal{Gamma}\).

It can then be shown that for any \(\mathcal{V}\)-structure \(M\), \(\mathcal{T}(M)\) \((M)\) is a complete \(\mathcal{V}\)-theory (for proof, see, e.g., [Hed04], p90).
Definition (PageIndex(13))

A set of sentences $\Gamma$ is said to be consistent if no contradiction can be derived from $\Gamma$.

Definition (PageIndex(14))

(Theory) A theory is a consistent set of sentences.

The latter two definitions are particularly relevant later on when we look at the typical reasoning services for ontologies.

Example (PageIndex(2)):

How does all this work out in practice? Let us take something quasi-familiar: a conceptual data model in Object-Role Modeling notation, depicted in the middle part of Figure 2.1.2, with the top-half its ‘verbalisation’ in a controlled natural language, and in the bottom-part some sample objects and the relations between them.

![Diagram of ORM notation, verbalization, and data in the database](https://eng.libretexts.org/Bookshelves/Computer_Science/Book%3A_An_Introduction_to_Ontology_Engineering_(Keet)/03%3A...)

Figure 2.1.2: A theory denoted in ORM notation, ORM verbalization, and some data in the database. See Example 2.1.2 for details.

First, we consider it as a theory, creating a logical reconstruction of the icons in the figure. There is one binary predicate, attends, and there are two unary predicates, Student and DegreeProgramme. The binary predicate is typed, i.e., its domain and range are defined to be those two entity types, hence:

\[
\forall x, y(\text{attends}(x, y) \rightarrow \text{Student}(x) \land \text{DegreeProgramme}(y))\]

Note that $x$ and $y$ quantify over the whole axiom (thanks to the brackets), hence, there are no free variables, hence, it is a sentence. There are two constraints in the figure: the blob and the line over part of the rectangle, and, textually, “Each Student attends exactly one DegreeProgramme” and “It is possible that more than one Student attends the same DegreeProgramme”. The first constraint can be formalised (in short-hand notation):
\[\forall x (\text{Student}(x) \rightarrow \exists^{=1} y \text{ attends}(x, y))\]

The second one is already covered with Eq. 2.1.12 (it does not introduce a new constraint). So, our vocabulary is \{\text{attends, Student, DegreeProgramme}\}, and we have two sentences (Eq. 2.1.12 and Eq. 2.1.13). The sentences form the theory, as they are not contradicting and admit a model.

Let us now consider the structure. We have a non-empty underlying set of objects: \((\Delta =) \{\text{John, Mary, Fabio, Claudia, Markus, Inge, ComputerScience, Biology, Design}\}\). The interpretation then maps the instances in \(\Delta\) with the elements in our vocabulary; that is, we end up with \{\text{John, Mary, Fabio, Claudia, Markus, Inge}\} as instances of \text{Student}, and similarly for \text{DegreeProgramme} and the binary \text{attends}. Observe that this structure does not contradict the constraints of our sentences.

---

**Equivalences**

With the syntax and semantics, several equivalencies between formulae can be proven. We list them for easy reference, with a few ‘informal readings’ for illustration. \(\phi\), \(\psi\), and \(\chi\) are formulas.

**Commutativity:**

\[\phi \land \psi \equiv \psi \land \phi\]
\[\phi \lor \psi \equiv \psi \lor \phi\]
\[\phi \leftrightarrow \psi \equiv \psi \leftrightarrow \phi\]

**Associativity:**

\[((\phi \land \psi) \land \chi) \equiv \phi \land (\psi \land \chi)\]
\[((\phi \lor \psi) \lor \chi) \equiv \phi \lor (\psi \lor \chi)\]

**Idempotence:**

\[\phi \land \phi \equiv \phi\]  // 'itself or itself is itself'
\[\phi \lor \phi \equiv \phi\]

**Absorption:**

\[\phi \land (\phi \lor \psi) \equiv \phi\]
\[\phi \lor (\phi \land \psi) \equiv \phi\]

**Distributivity:**

\[\phi \lor (\psi \land \chi) \equiv (\phi \lor \psi) \land (\phi \lor \chi)\]
\[\phi \land (\psi \lor \chi) \equiv (\phi \land \psi) \lor (\phi \land \chi)\]
Double Negation:
\(\neg\neg\phi\equiv\phi\)

De Morgan:
\(\neg(\phi\wedge\psi)\equiv\neg\phi\vee\neg\psi\)
\(\neg(\phi\vee\psi)\equiv\neg\phi\wedge\neg\psi\)  \(//\) ‘negation of a disjunction implies the negation of each of the disjuncts’

Implication:
\(\phi\to\psi\equiv\neg\phi\vee\psi\)

Tautology:
\(\phi\vee\top\equiv\top\)

Unsatisfiability:
\(\phi\wedge\bot\equiv\bot\)

Negation:
\(\phi\wedge\neg\phi\equiv\bot\)  \(//\) something cannot be both true and false
\(\phi\vee\neg\phi\equiv\top\)

Neutrality:
\(\phi\wedge\top\equiv\phi\)
\(\phi\vee\bot\equiv\phi\)

Quantifiers:
\(\neg\forall x.\phi\equiv\exists x.\neg\phi\)
\(\neg\exists x.\phi\equiv\forall x.\neg\phi\)  \(//\) ‘if there does not exist some, then there's always none’
\(\forall x.\phi\wedge\forall x.\psi\equiv\forall x.(\phi\wedge\psi)\) if \(x\) is not free in \(\psi\)
\(\forall x.\phi\vee\forall x.\psi\equiv\forall x.(\phi\vee\psi)\) if \(x\) is not free in \(\psi\)
\((\exists x.\phi)\wedge\psi\equiv\exists x.(\phi\wedge\psi))\) if \((x)\) is not free in \((\psi)\)

\((\exists x.\phi)\vee\psi\equiv\exists x.(\phi\vee\psi))\) if \((x)\) is not free in \((\psi)\)

Note \(\PageIndex{1}\):

The ones up to (but excluding) the quantifiers hold for both propositional logic and first order predicate logic.

---

**Footnotes**

1 or, for that matter, whether there is a reality and whether we have access to it

2 There are animals that indeed do not have 4 limbs (e.g., the sirens and the Mexican mole lizard have only two limbs and the millipede IIdacme plenipes has 750 (the most), and there are animals with specific numbers of limbs in between, but that is beside the point as it would not change much the features used in the formalisation, just become more cluttered.