1 First-Order Predicate Logic

1.1 Syntax

In this chapter, we'll review the syntax and semantics of first-order predicate logic. Unlike with propositional logic, most – some would even say: all – mathematical reasoning can be formalized in this language.

I'll begin with a simple version the language, without function symbols and identity. These will be added in section 1.4.

For the moment, then, the *primitive expressions* of a first-order language $\mathfrak{L}_1$ fall into the following kinds, whose instances must be distinct:

- an infinite set of *individual variables*,
- an infinite set of *individual constants*,
- for each natural number n, a set of *n-ary predicate symbols*,
- the *connectives* '$\neg$' and '$\rightarrow$',
- the *universal quantifier symbol* '$\forall$',
- the parentheses '(' and ')'.

The individual constants and predicate symbols are classified as non-logical.

Definition 1.1

The *singular terms* of a first-order language $\mathfrak{L}_1$ are its individual constants and its variables.
A *formula* of $\mathfrak{L}_1$ [later use $\mathfrak{L}_1^=$] is a finite string built up according to the following formation rules:

(i) If P is an n-ary predicate symbol and $t_1, \ldots, t_n$ are singular terms then $Pt_1 \ldots t_n$ is a formula.
(ii) If A is a formula, then so is $\neg A$.
(iii) If A and B are formulas, then so is $(A \rightarrow B)$.
(iv) If x is a variable and A is a formula then $\forall xA$ is a formula.

Here, 'P', 't_1', 't_n', 'A', 'B', 'x' are metalinguistic variables standing for expressions in the object language $\mathfrak{L}_1$. I haven't specified what the predicate symbols, individual constants, and variables of $\mathfrak{L}_1$ should look like. A popular convention is to use capital letters as predicate symbols, lower-case letters from the beginning of the alphabet as individual constants, and lower-case letters from the end of the alphabet (perhaps with numerical subscripts) as variables. With this convention, the following might be example formulas, assuming that 'F' is 1-ary (also called *monadic*) and 'R' 2-ary (or *binary*):

$$Fa,\ \ Rab,\ Rax,\ (\neg Fy \to Rax),\ (Fa \to \forall x(Fx \to Rax))$$

Exercise 1.1

Is $\forall x Fa$ a formula of $\mathfrak{L}_1$?

As in the case of propositional logic, we introduce some shortcuts in the metalanguage, writing

- $(A \wedge B)$ for $\neg(A \to \neg B)$;
- $(A \vee B)$ for $(\neg A \to B)$;
- $(A \leftrightarrow B)$ for $\neg((A \to B) \to \neg(B \to A))$.
- $\top$ for $A \to A$;
- $\bot$ for $\neg(A \to A)$;
- $\exists x A$ for $\neg\forall x \neg A$.

We'll omit parentheses when they are not needed for disambiguation.

Now a few words on how this kind of language is meant to be used.

Individual constants function as names. Each individual constant picks out an object – which might be a person, a number, a set, or whatever. Predicate letters express properties or relations. In a formal theory of arithmetic, for example, we might have individual constants '0' and '1' for the numbers 0 and 1, and a binary predicate 'G' that expresses the greater-than relation. '$G10$' is then a true formula, stating that 1 is greater than 0. '$G11$' falsely states that 1 is greater than itself.

The quantifier symbol '$\forall$' allows making general claims about all objects – or, more precisely, about all objects in the intended domain of discourse. In a theory of arithmetic, the intended domain of discourse would consist of the natural numbers 0, 1, 2, 3, etc. It would not include, say, Julius Caesar. Thus '$\forall x Gx0$ 'might state (falsely) that every natural number is greater than 0, while '$\forall x(Gx1 \to Gx0)$' states (truly) that every number that is greater than 1 is greater than 0. '$\exists x G1x$', which is short for '$\neg\forall x \neg G1x$', states (truly) that 1 is greater than some number.

Exercise 1.2

I said above that the primitive expressions must be "distinct". To be pediantic, we should say that none is part of another. Explain why it would be a problem to have '1', '2', ...

The quantificational constructions of first-order logic don't work like ordinary quantificational constructions in natural language, and it takes some practice to become fluent in their use. The closest translation of '$\forall x Gx0$' in English is something like

> Everything is such that it is greater than 0.

Of course, this can be simplified to 'everything is greater than 0', but here 'everything' combines directly with a predicate ('is greater than 0'), whereas the $\mathfrak{L}_1$-quantifier '$forallx$' combines with an expression of sentential type, with '$Gx0$', which means something like 'it is greater than 0'. The bound variable 'x' appears in the English translation only once, as the pronoun 'it'. Overt variables are rarely used in English, but they can be useful when quantifiers are nested:

> For every number x there is a number y greater than x such that every number greater than y is greater than x.

This can be easily expressed in $\mathfrak{L}_1$:

$$\forall x \exists y (Gyx \wedge \forall z (Gzy \rightarrow Gzx)).$$

[Exercise?]
Now consider the expression

$$Gx0.$$

According to definition **??**, it is a formula. But what does it say? I'm not asking how to pronounce it. We can, of course, read it as 'gee ex zero' or as 'x is greater than zero'. But 'x' doesn't pick out any definite number. Only individual constants pick out definite objects. Variables are devices to construct quantified statements. So '$Gx0$' doesn't really say anything. It is neither true nor false. (We might say that it is true "relative to some

interpretations of 'x' " and false relative to others. For example, it is true if we interpret 'x' as picking out 1, and false if we interpret it as picking out 0.)

The problem with '$Gx0$' is not that it contains a variable. '$\forall xGx0$' also contains a variable, but it raises no problems. On out arithmetical interpretation, it says that every number is greater than 0, which happens to be false. The problem with '$Gx0$' is that it contains a *free* variable – a variable that isn't bound by a quantifier.

Formulas with free variables are called *open*. Formulas with out free variables are *closed*. Only closed formulas make an outright claim about the intended domain of discourse. We allow for open formulas only because they allow for a simple specification of the construction rules for larger sentences.

Let's make these distinction more precise.

Definition 1.2

A *subformula* of a formula is a part of a formula that is itself a formula.
A *quantifier* consists of the symbol '$\forall$' followed by a variable. The variable is said to be *bound* by the quantifier.
The *scope* of an occurrence of a quantifier $\forall x$ in a formula is the shortest subformula that contains the occurrence.
An occurrence of a variable in a formula is *bound* if it lies in the scope of an occurrence of a quantifier that binds it.
An occurrence of a variable that isn't bound is *free*.
A formula in which some variable occurs free is *open*.
A formula in which no variable occurs free is *closed*.
A *sentence* is a closed formula.

Exercise 1.3

Why do I say that "occurrences" of a variable in a formula are free or bound? Why not simply say that a variable is free or bound?

Exercise 1.4

Which of these are open/closed? ...

Exercise 1.5

$\mathfrak{L}_1$ *extends* $\mathfrak{L}_0$ in the sense that every $\mathfrak{L}_0$-sentence is also an $\mathfrak{L}_1$-sentence. Explain why this is true.

The language of first-order logic is more restrictive than the formal language introduced by Frege. We only allow quantifying into the position of individual constants. Frege also allowed quantification into the position of predicates, as in $\forall XXa$. This kind of *second-order* quantification has interesting applications in mathematics and philosophy, but we'll only turn to it in chapter **??**.

1.2 The first-order predicate calculus

We're going to define a Hilbert-style proof calculus for $\mathfrak{L}_1$. We'll have the same axiom schemas A1-A3 as in the propositional calculus, and the same inference rule of Modus Ponens:

(A1)	$A \to (B \to A)$
(A2)	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
(A3)	$(\neg A \to \neg B) \to (B \to A)$
(MP)	From A and $A \to B$ one may infer B.

In addition, we need some principles for dealing with quantifiers. For example, we want to allow reasoning from the universal statement $\forall xFx$ to its instances like Fa and Fb. To state this in full generality, we need to define the notion of substitution.

Definition 1.3

If A is an $\mathfrak{L}_1$-formula, x a variable, and t a singular term, then $A(x/t)$ is the formula obtained from A by replacing all free occurrences of x in A with t.

The substitution operator '(x/t)' obviously belongs to our metalanguage, not to $\mathfrak{L}_1$ itself. For example, the metalinguistic expression '$Fx(x/a)$' denotes the object-language sentence 'Fa'. In informal discussions, I'll sometimes also write '$A(x)$' to indicate that A is a formula in which the variable x occurs freely, in which case '$A(t)$' is shorthand for '$A(x/t)$'.

Exercise 1.6

Let A be $\forall x(Fx \to Gy) \to \forall yFy$. What is $A(x/a)$?

The general rule of "Universal Instantiation" says that one may reason from $\forall xA$ to $A(x/c)$, where c is an individual constant. In Hilbert-style axiomatic calculi, we try to minimize primitive rules. So instead of adding a new inference rule, we'll add a corresponding axiom schema:

(A4) $\forall xA \to A(x/c)$.

With this, we can reason from $\forall xA$ to $A(x/c)$ by Modus Ponens.

We'll introduce a genuine rule – called /(Universal) Generalization/ – for the converse inference, from $A(x/c)$ to $\forall xA$.

(Gen) From $A(x/c)$ one may infer $\forall xA$.

This rule is obviously unsound when reasoning from assumptions: from the assumption that Joan sleeps we can't infer that everyone sleeps. But remember that each line in a (Hilbert-style) axiomatic proof is either an axiom or follows from an axiom by an inference rule. None of our axioms (or rules) will make specific claims about a particular object c that it wouldn't make about all objects. Thus if $A(c)$ can be derived from the axioms then one could equally derive $A(d)$ for any other constant d.

To get a complete calculus, we need one more axiom:

(A5) $\forall x(A \to B) \to (A \to \forall xB)$, if x is not free in A.

To see the point of this, suppose that in the course of a proof we have established the following claims, for some A and $B(x)$, where x isn't free in A:

$\forall x(A \to B(x))$
A

If we had a rule of Universal Instantiation, we could deduce $A \to B(c)$ from the first line, then use Modus Ponens to infer $B(c)$ and finally infer $\forall xB(x)$ by Universal Generalization. It turns out that this reasoning can't be replicated after we've replace Universal

Instantiation by the axiom schema (A4). So we add (A5), which allows inferring $\forall x B(x)$ by two applications of Modus Ponens.

I'll use '$\vdash A$' to express that A is provable in the first-order predicate calculus. That is: A can be derived from some instances of A1-A5 by MP and Gen.

As in the case of propositional logic, it's convenient to have a more general notion of proof that covers deductions from premises.

Definition 1.4

A (first-order) *deduction* from a set Γ of sentences is a finite list of sentences each of which is either an axiom, an element of Γ, or follows from previous sentences by MP or Gen – but without applying Gen to individual constants c that occur in Γ.

A deduction *of* a sentence A from a set Γ of sentences is a deduction from Γ whose last element is A.

I've already mentioned that the Gen rule is not generally sound for deductions from premises: we don't want to infer $\forall x Fx$ from Fa. In a deduction, the individual constant c that is turned into a universally bound variable must not occur in any premise.

Let's show that, as in the propositional calculus, the notion of deduction reduces to our original notion of proof:

Deduction Theorem

If $\Gamma, A \vdash B$ then $\Gamma \vdash A \to B$.

> The proof is almost exactly like in the previous chapter. We only need an extra clause for applications of the rule (Gen), where we appeal to the axiom schema A5.
>
> See e.g. Bostock 224

Given Gen and the DT, we can derive A5, so we won't need to look at that again. (Bostock 224f., probably best to use the proof from p.225)

From A4 and the DT, we obviously get the rule of Universal Instantiation, or $\forall$ Elimination:

Universal Instantiation

f $\Gamma \vdash \forall x A$ then $\Gamma \vdash A(x/c)$ for any individual constant c.

Now we don't need A4 any more.

Since we have the DT, the proofs of Reductio, Ex Falso, DNE, and PIP from the previous chapter all go through unchanged. We also have the structural principles Assumptions, Monotonicity, and Cut.

In fact, we know from completeness theorem that all truth-functional tautologies are provable from P1-P3 and MP. So we have this fact:

$\Gamma \vdash A$ whenever A is a tautology.

This could be considered a derived rule. It's an odd rule. But it's verifiable: there is a mechanical way to check if something is a tautology; e.g. with truth tables.

In the previous chapter, I should that our results about deduction can be turned into different types of calculi. This is still true for first-order logic. I won't stop to explain it again.

Exercise: show that if $\Gamma \vdash A(x/c)$ then $\Gamma \vdash \exists x A$.

Side note: I have stipulated that each line in a proof must be a sentence. In some versions of the first-order calculus, open formulas are allowed in proofs: from $\forall x Fx$, one may infer Fx; the variable indicates that x is an "arbitrary individual". The same *sentences* are provable in either version of the calculus. Since individual constants aren't used in reasoning from universal formulas, the alternative calculus doesn't need my assumption that $\mathfrak{L}_1$ has an unbounded supply of such constants. End of side note.

1.3 Semantics

A language is not just a system of meaningless sounds and scribbles, combined according to seemingly arbitrary syntactic rules. If we want to use a first-order language $\mathfrak{L}_1$ to formalize statements about this or that subject matter, we need to give meaning to its non-logical vocabulary.

On the truth-conditional approach, sentence meanings are understood in terms of truth-conditions: the meaning of a sentence should settle, for each conceivable scenario, whether the sentence is true or false (in that scenario). The meaning of sub-sentential expressions is determined by their contribution to sentence meaning. Let's think about what this implies for the expressions of $\mathfrak{L}_1$, focusing on simple atomic sentences like 'Fa'.

These are meant to function much like subject-predicate sentences in natural language: 'Fa' might be a formal way of saying 'Kurt Gödel is a logician'. In the English sentence, the name 'Kurt Gödel' picks out an individual, and the predicate 'is a logician' attributes

a property to that individual. This suggests that we could fix the truth-conditions of atomic $\mathfrak{L}_1$-sentences by assigning

- a (definite, precise) object to each individual constant, and - a (definite, precise) property or relation to each predicate letter.

We don't need to worry much about the concept of an "object". We'll assume that in any use of a first-order language, there is an intended domain of things we want to talk about; each such thing will qualify as an "object" – as a candidate denotation of individual constants. But what is a property (or relation)?

The standard approach in the model theory of first-order logic is to represent a property by its *extension*: by the sets of objects that have the property. The property of being a logician, for example, would be identified with the set { Kurt Gödel, Gottlob Frege, Rózsa Péter, …} of all logicians.

As a general theory of meaning, this is clearly inadequate. After all, there are conceivable scenarios in which Kurt Gödel is not a logician, or in which, say, Carla Bruni is a logician; but there is no conceivable scenario in which Gödel isn't in the set $\{KurtGdel, \ldots\}$ or in which Carla Bruni is in that set. There are two reasons why we can nonetheless largely get away with the "extensional" account of predicate meaning.

The first is that many logicians are mostly interested in mathematical interpretations, where we might read 'a' as denoting the number 2 and 'F' as expressing the property of being prime. While Kurt Gödel could have failed to be a logician, the number 2 could not have failed to be prime: there is no conceivable scenario in which 2 isn't prime; nor is there a conceivable scenario in which, say, the number 4 is prime. In general, it is reasonable to hold that purely mathematical truths don't vary across conceivable scenarios. In this special case, truth-conditions are determined by extensions.

The second reason why we can get away with an extensional construal of properties is analogous to the reason why we can get away with modelling sentence meanings as truth-values in propositional logic.

Recall our intuitive gloss on logical entailment: some premises logically entail some conclusion if there is no conceivable scenario in which the premises are true and the conclusion false, under any interpretation of the non-logical vocabulary; equivalently: – if there is no pair of a scenario and an interpretation that makes the premises true and the conclusion false. Now, what do you need to know about a scenario S and an interpretation I in order to figure out whether an $\mathfrak{L}_1$-sentence like Fa is true or false (in S under I)?

It would suffice to know (1) what object is picked out by 'a' under I, (2) what property is expressed by 'F' under I, and (3) whether that object has that property in S. But you don't need all that information. It would also suffice to know (1) what object is picked

out by 'a' under I, and (2) which objects in S have the property expressed by 'F' under I. For example, if I tell you that 'a' picks out Kurt Gödel and that 'F' expresses a property that belongs to Kurt Gödel and Carla Bruni (in S), you'll know enough to figure out that 'Fa' is true – although you don't really know what the sentence says. That is, if we're interested in interpretations only for the purpose of developing a precise account of (first-order) logical entailment, we can ignore any aspect of predicate meaning that goes beyond extension.

Definition 1.5

A *model* M of a first-order language $\mathfrak{L}_1$ consists of

(i) a non-empty set D, called the *domain* or *universe* of M, and
(ii) a function ι that assigns
- to each individual constant of $\mathfrak{L}_1$ an object in D, and
- to each n-ary predicate of $\mathfrak{L}_1$ a set of n-tuples of objects in D.

(Models are often called 'interpretations', but outside pure mathematics this is a dubious label, for the reasons given above.)

Again, the domain of a model is assumed to comprise all the objects one intends to speak about. The domain can be infinite, but it can't be empty. Each individual constant is associated with an object in the domain. Different constants can pick out the same object. (We need at least one object in the domain, as otherwise we couldn't interpret the names.) Each predicate symbol is associated with an n-tuple of objects from the domain. An *n-tuple* is a list of n objects. A 1-tuple is simply an object. So a "set of 1-tuples of objects" is simply a set of objects; a set of 2-tuples of objects is a set of pairs of objects, and so on.

Exercise: what about zero-ary predicates? can you make this work?

Now we can define what it takes for a sentence A to be true in a model M – for short, $M \models A$. Most of this is easy. For example, an atomic sentence Fa is true in a model M iff the object assigned (by M) to 'a' belongs to the set assigned to 'F'. The clauses for $\neg$ and $\rightarrow$ remain as in the previous section. The only difficulty arises with quantified sentences. Let $A(x)$ be some formula in which x is free. Under what conditions is $\forall x A(x)$ true in a model M?

As a first shot, we might suggest that $\forall x A(x)$ is true iff $A(c)$ is true for every individual constant c. (This is called a *substitutional interpretation* of the quantifier.) But what if some objects in the intended domain don't have a name? Most real numbers, for example,

are not denoted by any English expression (because there are strictly more real numbers than English expressions), and we generally want to allow for this kind of situation in $\mathfrak{L}_1$

There are two common ways to proceed. The first is to declare that $\forall x A(x)$ is true in a model M iff $A(x)$ is true *for every way of assigning an individual to x*. This requires extending the definition of truth in a model so that open formulas are true or false in a model M relative to an *assignment function* g that assigns an object to each variable.

The second option is to declare that $\forall x A(x)$ is true in M iff $A(c)$ is true in every variant of the model M that differs from M at most in the object it assigns to c, where c is some individual constant that doesn't occur in $A(x)$.

The two options amount to the same thing. I'll use the second. Here, then, is the full definition of truth in a model.

Definition 1.6

An $\mathfrak{L}_1$-sentence A is true in a model M (for short, $M \models A$) if one of the following conditions holds.

1. A is an atomic sentence $Pc_1 \dots c_n$ and $(\imath_M(c_1), \dots, \imath_M(c_n)) \in \imath_M(P)$.
2. A is of the form $\neg B$ and $M \not\models B$.
3. A is of the form $(B \to C)$ and $M \not\models B$ or $M \models C$.
4. A is of the form $\forall x B$ and $M' \models B(x/c)$ for every model M' that differs from M at most in the object assigned to c, where c is the alphabetically first individual constant that does not occur in B.

We can now define entailment. Every model of Γ is a model of A.

A and B are logically equivalent iff each entails the other.

Exercise 1.7

Show that $\Gamma, A \models B$ iff $\Gamma \models A \to B$.

Exercise 1.8

Show that $\forall x(A \to B) \models \forall x A \to \forall x B$. Show that if x is not free in B then $\forall x(A \to B)$ is equivalent to $\exists x A \to B$.

Exercise 1.9

Show that substituting a subformula by an equivalent subformula results in an equivalent formula.

Definition 1.7: Prenex Normal Form

A formula is in *prenex normal form* if all its quantifiers are at the beginning.

The idea of the proof is to demonstrate that a quantifier which occurs somewhere in the middle of a formula can always be moved one step to the left, and by sufficiently many such steps we can bring all the quantifiers as far to the left as possible, so that they do all occur in a block at the beginning.

All we need to show is that

- A $\to$ xB is equivalent to x(A $\to$ B), if x is not free in A.
- A $\to \neg$ xB is equivalent to $\neg$ x(A $\to$ B), if x is not free in A.
- alpha-equivalence

Maybe show semantically?

Needed to generalize Σ and Π: Call a formula of LPA essentially Σ_n or Π_n if it can be transformed into a Σ_n or a Π_n formula, respectively, by the usual prenexing rules.

1.4 Functions, identity, and mathematical structures

For mathematical purposes, it is useful to have a version of predicate logic with function symbols and identity. Consider the statement

$$1 + 2 = 3.$$

How could we formalize this? In $\mathfrak{L}_1$, we would have to use a three-place predicate symbol S and write

$$S(1, 2, 3).$$

But this obscures the real structure of the statement, which states an identity between 1 + 2 and 3.

The first step to remedy this situation is introduce a predicate for identity. We'll use '=' for this predicate. (We also use '=' for identity in our metalanguage; you'll have to figure out from context which is in play.) Officially, all predicates are placed in front of their arguments, so we should write '$= ab$'. In practice, we'll "abbreviate" this as '$a = b$'. We'll also write '$a \neq b$' for '$\neg = ab$'.

Exercise 1.10

Construct a sentence with '=' as the only predicate symbol that is true in a model M only if the domain of M has (a) at least two members, (b) at most two members, (c) exactly two members.

Of course, nothing stopped us from having a predicate for identity before. The real novelty is that we'll classify '=' as a *logical* expression, like '$\wedge$' and '$\forall$' and unlike 'a' and 'F'. This means that we'll have special rules for reasoning with '=', and that models are not allowed to interpret '=' as meaning anything other than identity.

Before we get to this, I want to introduce the second addition to $\mathfrak{L}_1$ that will allow us to form complex singular terms like '1 + 2'.

The expression '1 + 2' denotes a number: the number 3. (That's why '1+2 = 3' is true.) How does this come about? We can understand the '+' sign as expressing a function that maps a pair of numbers to a number: 1 and 2 to 3, 2 and 5 to 7, and so on. So understood, '+' is a *function symbol*. Such symbols are ubiquitous in maths, and we'll add them to $\mathfrak{L}_1$.

First, however, a few general words on the concept of a function, which will play an important role throughout these notes.

A function, in the mathematical and logical sense, takes one or more objects of a certain kind as input and (typically) returns an object of a certain kind as output. An input to a function is also called an *argument* to the function; the output is called the function's *value* for that argument.

The addition function takes two numbers as input and returns a number. The square function takes a single number and returns a number (the square of the input). The inputs and outputs don't need to be numbers. There is an "area" function that takes a country as input and returns its size in (say) square kilometres. And there is a "mother" function that takes a person as input and returns their mother.

The class of inputs to a function is called *domain*; the class of candidate outputs are

its *codomain*. If a function has domain X and codomain Y, we say that it is a function *from X to Y*. If all inputs and outputs of a function belong to a set X, we say that it is a function *on X*.

Some functions are not defined for all objects in their domain. The division function, for example, takes two numbers as input and returns a number, but it is undefined if the second input number is zero. Such functions are called *partial*.

Functions are often associated with a recipe or algorithm for determining the output for a given input. (There are algorithms for computing sums or squares.) But we don't build this into the concept of a function. Any mapping from inputs to outputs is a function, even if there is no recipe for determining the output.

Since functions are just mappings from inputs to outputs, they are fully determined by their values for each input. Consider, for example, the function g from numbers to numbers that takes a number x and as input and returns x^2 if Goldbach's conjecture is true and 0 if Goldbach's conjecture is false. Goldbach's conjecture says that every even number greater than 2 is the sum of two primes. It is not known whether the conjecture is true. So we don't know what g returns for inputs other than 0. But we know that g is either identical to the square function or to the constant function that returns 0 for every input.

Exercise 1.11

Give another example of a function with 1, 2, 3 arguments.

Now we'll add function symbols and an identity predicate to our first-order language $\mathfrak{L}_1$. Let's call the resulting language $\mathfrak{L}_1^=$. Function symbols combine with singular terms to form new singular terms. For example, if f is a two-place function symbol and a, b are individual constants, then $f(a, b)$ is a singular term; it denotes the value of the function f for the arguments a and b.

Definition 1.8

A *(singular) term* of $\mathfrak{L}_1^=$ is a finite string constructed by the following rules.

- Every variable and individual constant is a singular term.
- If f is an n-ary function symbol ($n > 0$) and $t_1, \dots, t_n$ are singular terms then $f(t_1, \dots, t_n)$ is a singular term.

By '$f(t_1, \dots, t_n)$' I mean the string that begins with f, followed by the opening parenthesis, followed by the terms $t_1, \dots, t_n$ separated by commas, and ending with a closing parenthesis.

We could allow for zero-place function symbols. A (total) zero-ary function takes nothing as input and returns a single value; a zero-place function symbol f simply denotes this value; it behaves just like a name.

A 0-ary function symbol functions syntactically and semantically just like an individual constant.

Officially, function symbols are placed in front of their arguments, with parentheses and commas to separate the arguments. $(x \times y) + z$ would be written $+(\times(x, y), z)$. For the sake of readability, we'll allow the more familiar infix notation as a metalinguistic "abbreviations".

Exercise 1.12

Consider a language of arithmetic with constants 0,1,2,3..., and + and × intended to mean addition and multiplication. Can you write the statement of Lagrange's Theorem, which states that every natural number is the sum of four squares?

Exercise 1.13

Suppose we used infix notation without parenthesis, writing, for example, x+y for the sum of x and y. This would cause a problem with more complex terms that contain functional terms. Can you explain the problem? [Think of subtraction: x-y-z has two readings. We want *unique readability*.]

The definition of formulas is the same as above.

Definition 1.9

A *formula* of $\mathfrak{L}_1^=$ is a finite string built up according to the following formation rules:

(i) If P is an n-ary predicate symbol and $t_1, \dots, t_n$ are singular terms then $Pt_1 \dots t_n$ is a formula.
(ii) If A is a formula, then so is $\neg A$.
(iii) If A and B are formulas, then so is $(A \to B)$.
(iv) If x is a variable and A is a formula then $\forall x A$ is a formula.

The axiomatic calculus can stay the same, except that we now allow for closed terms wherever we previously appealed to individual constants. That is, we have

(A4) $\forall x A \to A(x/t)$

(Gen) From $A(x/t)$ infer $\forall x A$.

We'll add some new axiom schemas for identity: for any terms t_1 and t_2 and formulas A in which only x is free,

(SI) $t_1 = t_2$for any closed terms t_1 and t_2

(LL) $t_1 = t_2 \to (A(x/t_1) \to A(x/t_2))$

The somewhat strange formulation of LL is a trick that allows replacing only some occurrences of t_1 by t_2.

Let's think about models.

Singular terms don't make a claim; they merely serve to pick out some object.

We define $[t]^M$ so that $[f(t_1, \dots, t_n)]^M = [f]^M([t_1]^M, \dots, [t_n]^M)$.

In the model definition, we require function symbols to denote total functions on the domain of discourse D. Otherwise we'll have empty terms.

Definition 1.10

A *model* M of a first-order language $\mathfrak{L}_1^=$ consists of

(i) a non-empty set D, called the *domain* or *universe* of M, and

(ii) a function $\imath$ that assigns

- to each individual constant of $\mathfrak{L}_1$ an object in D,
- to each n-ary function symbol of $\mathfrak{L}_1^=$ an n-ary total function on D, and
- to each non-logical n-ary predicate of $\mathfrak{L}_1$ a set of n-tuples of objects in D.

Definition 1.11

An $\mathfrak{L}_1^=$-sentence A is true in a model M (for short, $M \models A$) if one of the following conditions holds.

1. A is an identity sentence $t_1 = t_2$ and $[t_1]^M = [t_2]^M$.
2. A is any other atomic sentence $Pt_1 \dots t_n$ and $([t_1]^M, \dots,) \in \imath_M(P)$.
3. A is of the form $\neg B$ and $M \not\models B$.
4. A is of the form $(B \to C)$ and $M \not\models B$ or $M \models C$.
5. A is of the form $\forall x B$ and $M' \models B(x/c)$ for every model M' that differs from M at most in the object assigned to c, where c is the alphabetically first individual constant that does not occur in B.

Exercise 1.14

Explain why $\models a = a$, if a is an individual constant.

Exercise 1.15

Define a model in which 1+1=2 is true and one in which it is false.

Once we have function symbols, the assumption that singular terms of our formal language must not be empty becomes even more problematic than it was before. As I said, in ordinary maths, we have expressions like $1/0$ that don't denote anything. We could allow for empty terms in our models. This would have the advantage that we could then also allow for empty domains. But we'd have to change our calculus. For example, we shouldn't allow the inference from $\forall x Fx$ to Fa to $\exists x Fx$. There are different ways of pursuing this approach. They are studied in a non-classical predicate logic called *free logic*. For most applications we prefer to keep things simple.

Many important mathematical structures consist of a set of objects together with some operations and relations on these objects, and sometimes with a designated object playing a special role. For example, we might construe the structure of natural numbers as consisting of the set $\mathbb{N}$ of numbers 0,1,2,…, the designated object 0, the addition operation $+$, the multiplication operation $\times$, and the less-than relation $<$. It is common to package all this into a list: $(\mathbb{N}, 0, +, \times, <)$.

When we talk about this structure in a first-order language, we'll naturally use $\mathbb{N}$ as the intended domain of discourse, we'll have an individual constant for 0, function symbols intended to express $+$ and $\times$, and a predicate symbol for $<$. In effect, the structure

$(\mathbb{N}, 0, +, \times, <)$ then provides a model for our language. That's why textbooks on mathematical logic often identify models with mathematical structures.

Careful: when specifying the algebra, we don't use the first-order object language. So '+' here is metalanguage. The object language may also have a '+' symbol or a '0' symbol, but they are not treated as logical. So their meaning isn't fixed. When we talk about logical entailment, we need to consider any possible way of interpreting '0' and '+'.

E.g, D = { Paris, Rome, Canberra }, +(x,y) = Paris, ×(x,y) = Rome, 0 = Canberra.

Note that, technically, we don't allow for languages with only one or two individual constants. We need an unending supply of constants for two reasons. One, the interpretation of $\forall xA$ appeals to $A(x/c)$, where c is a constant that doesn't occur in A. The interpretation of $\forall x \forall yA$ therefore involves two constants, and so on. But it doesn't matter what M assigns to these further constants. This is also true for the other reason why we need extra constants: in our calculus, we often need to reason from $\forall xA(x)$ to $A(c)$, where c is a new constant, so that if we have derived $B(c)$ from $A(c)$, we can later apply Gen to get $\forall xB(x)$. Here, too, the extra constants only serve a Hilfsfunktion. They aren't used to pick out a particular object. (Such constants are historically called *eigenvariables*.)

Definition 1.12: Isomorphic Structures

Structures $\mathscr{A}$ and $\mathscr{B}$ are *isomorphic* if there is a bijection f from the domain of $\mathscr{A}$ to the domain of $\mathscr{B}$ such that xxx

Clearly, truth at a structure is preserved under isomorphisms. A full proof is in BBJ pp.140ff. Might move to next chapter?

Exercise 1.16

Show that isomorphisms are equivalence relations

1.5 Truth

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added to the final page this extra page has been added to receive it.

If you rerun the document (without altering it) this surplus page will go away, because LaTeX now knows how many pages to expect for this document.