

# Logic(s) for Computer Science - Week 8

## The Syntax of First-Order Logic

Ștefan Ciobâcă

December 12, 2017

### 1 Motivation and introduction

First-order logic, what we will be studying next, is an extension of propositional logic, extension that brings more expressivity. The additional expressivity is necessary in order to model certain statements that cannot be expressed in propositional logic, but that appear in practice.

In propositional logic, we cannot express naturally the following statement: *Any human is mortal*.

To model a statement in propositional logic, we identify the atomic propositions. Then we associate to each atomic proposition a propositional variable. The atomic propositions are the propositions that cannot be split into one or more smaller propositions, linked among them by the logical connectives of propositional logic: *and*, *or*, *not*, *implies* and *if and only if*.

We notice that the statement *Any human is mortal* cannot be decomposed into smaller statements linked among them by the logical connectives of propositional logic, as is described above. Therefore, in propositional logic, the statement is atomic. So we associate to the entire statement a propositional variable  $p \in A$ .

Let us now model the statement *Socrates is human*. Obviously, to this second statement we must associate another propositional variable  $q \in A$ . Let us assume that  $p$  and  $q$  are true. Formally, we work in a truth assignment  $\tau : A \rightarrow B$  where  $S(p) = 1$  and  $S(q) = 1$ . Can we draw the conclusion that *Socrates is mortal* in the truth assignment  $\tau$ ?

No, because to the statement *Socrates is mortal* we should associate a third propositional variable  $r \in A$ . We cannot draw any conclusion on  $S(r)$  from  $S(p) = 1$  and  $S(q) = 1$ . So, from the semantics of propositional logic, we cannot draw the conclusion that  $r$  is true in any truth assignment that makes both  $p$  and  $q$  true. This is despite the fact that, in any world where any human is mortal and Socrates is human, we can draw the conclusion that Socrates is mortal without failure. This difference between reality and our modelling indicates that our modelling is not sufficient for our purposes.

First-order logic includes, in addition to propositional logic, the notion of *quantifier* and the notion of *predicate*. The universal quantifier is denoted by  $\forall$

and the existential quantifier is denoted by  $\exists$ .

A predicate is a statement whose truth value depends on zero or more parameters. For example, for the statements above, we will be using two predicates: *Human* and *Mortal*. The predicate *Human* is the predicate that denotes the quality of being a human:  $Human(x)$  is true iff  $x$  is a human. The predicate *Mortal* is true when its argument is mortal. As the predicates above have only one argument/parameter, they are called *unary* predicates. Predicates generalize propositional variables by the fact that they can take arguments.

In this way, the statement *any human is mortal* will be modelled by the formula

$$\forall x.(Human(x) \rightarrow Mortal(x)),$$

which is read as follows: *for any  $x$ , if  $Human$  of  $x$ , then  $Mortal$  of  $x$* . The statement *Socrates is human* shall be modelled by the formula  $Human(s)$ , where  $s$  is a *constant* that denotes Socrates, just like 0 denotes the natural number zero. For example,  $Human(s)$  is true (as  $s$  stands for a particular human being – Socrates), by  $Human(l)$  is false if  $l$  is a constant standing for the dog *Lassie*.

The statement *Socrates is mortal* shall be represented by  $Mortal(s)$  (recall that the constant  $s$  stands for Socrates). The statement  $Mortal(s)$  is true, as Socrates is mortal; likewise, the statement  $Mortal(l)$  is also true.

We shall see that in first-order logic, the formula  $Mortal(s)$  is a logical consequence of the formulae  $\forall x.(Human(x) \rightarrow Mortal(x))$  and respectively  $Human(s)$ . Therefore, first-order logic is sufficiently expressive to explain theoretically the argument by which we deduce that *Socrates is mortal* from the facts that *Any human is mortal* and *Socrates is human*.

## 2 Structures and signatures

You have certainly met already several first-order logic formulae, without necessarily knowing that you are dealing with first-order logic. Consider the following formula:

$$\varphi = \forall x.\forall y.(x < y \rightarrow \exists z.(x < z \wedge z < y)).$$

The formula makes use of a binary predicate,  $<$ , that is defined as follows:  $<(x, y)$  is true if  $x$  is strictly smaller than  $y$ . In order to simplify our writing, we use the infix notation  $(x < y)$  instead of the prefixed notation  $(<(x, y))$  for many binary predicates (including for  $<$ ).

If the formula  $\varphi$  above is true? The formula states that between any two values of the variables  $x, y$  there is a third value, of the variable  $z$ . The formula is true if the domain of the variables  $x, y, z$  is  $\mathbb{R}$ , but it is false if the domain is  $\mathbb{N}$  (between any two real numbers there exists a third, but between two consecutive naturals there is no other natural number).

Generally, first-order formula refer to a particular *mathematical structure*.

**Definition 2.1.** A mathematical structure is a tuple

$$(D, R_1, \dots, R_n, f_1, \dots, f_m),$$

where:

- $D$  is a non-empty set called the domain of the structure;
- $R_i$  ( $1 \leq i \leq n$ ) are predicates (each of a certain arity) over the set  $D$ ;
- $f_i$  ( $1 \leq i \leq m$ ) are functions (each of a certain arity) over the set  $D$ .

Functions and predicates over  $D$

**Definition 2.2.** A function of arity  $n$  over the set  $D$  is any function

$$f : D^n \rightarrow D.$$

**Definition 2.3.** A predicate of arity  $n$  over the set  $D$  is any function

$$p : D^n \rightarrow B,$$

where  $B = \{0, 1\}$  is the usual set of truth values.

Here are a few examples of mathematical structures:

1.  $(\mathbb{N}, <, =, +, 0, 1)$ ;

The domain of the structure is the set of naturals. The structure contains two predicates:  $<$  and  $=$ , both of arity 2. The predicate  $<$  is the *smaller than* predicate on naturals, and the predicate  $=$  is the *equality* predicate over natural numbers.

The structure also contains three functions. The binary function  $+$  :  $\mathbb{N}^2 \rightarrow \mathbb{N}$  is the addition function for naturals, and the functions  $0$  :  $\mathbb{N}^0 \rightarrow \mathbb{N}$  and respectively  $1$  :  $\mathbb{N}^0 \rightarrow \mathbb{N}$  are the arity 0 functions (also called constant functions or simply constants) 0 and respectively 1.

Arity 0 functions

A function of arity 0 is also called a *constant*. According to our definition, such a function is of the form

$$c : D^0 \rightarrow D.$$

In general, the set  $D^n$  is the cartesian product of the set  $D$  with itself  $n$  times. Formally, we have that

$$D^n = \{(d_1, \dots, d_n) \mid d_1 \in D, \dots, d_n \in D\}.$$

In the particular case where  $n = 0$ , the set  $D^0$  has a single element,

namely the empty tuple, denoted  $()$ . For this reason, arity 0 functions receive no argument and return the same result every time. Practically for any constant  $c \in D$ , there exists a unique function  $: D^0 \rightarrow D$  that returns  $c$ . We will identify from hereon this function and the constant  $c$ .

2.  $(\mathbb{R}, <, =, +, -, 0, 1)$ ;

This structure contains two binary predicates,  $<$  and  $=$ , as well as three functions over  $\mathbb{R}$ : the binary function  $+$ , the unary function  $-$  (unary minus) and the constants  $0, 1 \in \mathbb{R}$ .

3.  $(\mathbb{Z}, <, =, +, -, 0, 1)$ ;

This structure is similar to that above, but the domain is the set of integers.

4.  $(B, \cdot, +, -)$ ;

This structure is a boolean algebra, where the domain is the set truth values and the functions are those that we studied in the first half of the semester. Such structures, without any predicates, are called *algebraic structures*.

5.  $(\mathbb{R}, <)$ .

This structure contains only a predicate of arity 2 (the *less than* relation over  $\mathbb{R}$ ) and no function. Structures without functions are called relational structures. Relational structures with a finite domain are called relational data bases and you will study them in your second year.

Let us go back to the earlier formula:

$$\varphi = \forall x. \forall y. (x < y \rightarrow \exists z. (x < z \wedge z < y)).$$

We have that this formula is true in the structure  $(\mathbb{R}, <, =, +, -, 0, 1)$  (between any two distinct real numbers there is another real number), but it is false in the structure  $(\mathbb{Z}, <, =, +, -, 0, 1)$  (because it is not true that between any two distinct integers there is a third integer – for example there is no such integer between two consecutive integers).

When we have a first-order formula and we wish to evaluate its truth value, we must fix the structure in which we work.

It is possible for two different structure to have a set of predicates and a set of functions with the same names. For example, the structures above,  $(\mathbb{R}, <, =, +, -, 0, 1)$  and respectively  $(\mathbb{Z}, <, =, +, -, 0, 1)$ . Even if the predicate  $<: \mathbb{R}^2 \rightarrow \mathbb{R}$  is different from the predicate  $<: \mathbb{Z}^2 \rightarrow \mathbb{Z}$ , they both have the same name:  $<$ .

Generally, in Mathematics and in Computer Science, we do not make any difference between a predicate and its name or between a function and its name. However, in Logic, the difference is extremely important. In particular, if we

refer to the name of a function, we shall use the phrase “functional symbol” (i.e., symbol standing for a function). When we refer to the name of a predicate, we shall use the phrase “predicate symbol” (or “relational symbol”). Why is the difference between a predicate and a predicate symbol important? Because we shall need to associate to the same predicate symbol several predicates, similarly to how we can associate several values to a program variable in an imperative language.

When we are interested only in the function and predicate names (not the function or predicates themselves), we work with signatures:

**Definition 2.4.** A signature  $\Sigma$  is a tuple  $\Sigma = (R_1, \dots, R_n, f_1, \dots, f_m)$ , where  $R_1, \dots, R_n$  are predicate symbols and  $f_1, \dots, f_m$  are functional symbols, with the property that each predicate symbol  $R_i$  ( $1 \leq i \leq n$ ) and each functional symbol  $f_i$  ( $1 \leq i \leq m$ ) has an associated natural number called its arity. We will denote by  $ar(f_i)$  the arity of the functional symbol  $f_i$  ( $1 \leq i \leq m$ ) and by  $ar(R_i)$  the arity of the predicate symbol  $R_i$  ( $1 \leq i \leq n$ ).

To a signature we can associate many structures:

**Definition 2.5.** If  $\Sigma = (R_1, \dots, R_n, f_1, \dots, f_m)$  is a signature, a  $\Sigma$ -structure is any structure  $S = (D, R_1^S, \dots, R_n^S, f_1^S, \dots, f_m^S)$  so that  $R_i^S$  is a predicate over  $D$  of arity  $ar(R_i)$ , and  $f_i^S$  is a function over  $D$  of arity  $ar(f_i)$ .

To remember!

Structure = domain + predicates + functions

Signature = predicate symbols + functional symbols

To a signature  $\Sigma$  we can associate many structures, which are called  $\Sigma$ -structures.

**Example 2.1.** Let  $\Sigma = (P, Q, f, i, a, b)$ , where  $P, Q$  are predicate symbols of arity  $ar(P) = ar(Q) = 2$  and  $f, i, a, b$  are function symbols having the following arities:  $ar(f) = 2$ ,  $ar(i) = 1$  and  $ar(a) = ar(b) = 0$ .

We have that  $(\mathbb{R}, <, =, +, -, 0, 1)$  and respectively  $(\mathbb{Z}, <, =, +, -, 0, 1)$  are  $\Sigma$ -structures.

After having fixed a signature  $\Sigma = (R_1, \dots, R_n, f_1, \dots, f_m)$ , we denote by  $\mathcal{P} = \{R_1, \dots, R_n\}$  the set of predicate symbols in the signature, and by  $\mathcal{F} = \{f_1, \dots, f_m\}$  the set of functional symbols in the signature. The set of predicate symbols of arity  $n$  is  $\mathcal{P}_n = \{P \mid ar(P) = n\}$ , and the set of functional symbols of arity  $n$  is  $\mathcal{F}_n = \{f \mid ar(f) = n\}$ .

### 3 The Syntax of First-Order Logic

Next, we shall study the syntax of first-order logic formulae (the mathematical definition of the way we write formulae, i.e. what are formulae). Then we will study the semantics of first-order formulae (how to compute the truth value of a formula).

A difference to propositional logic is that there are several first-order logic languages, one first-order language for each signature  $\Sigma$ . In propositional logic, there was just one language, *PL*.

Next, we shall fix a signature  $\Sigma$  that contains the predicate symbols in  $\mathcal{P}$  and the functional symbols in  $\mathcal{F}$ .

We recall that the elements of  $\mathcal{F}$ , called functional symbols, have an associated natural number called the arity. By  $\mathcal{F}_n$  we denote the set of functional symbols of arity  $n$ . The elements of  $\mathcal{F}$  will usually be denoted by  $f, g, h, f', g', f_1, g_2, h''$ , etc. The elements of  $\mathcal{F}_0$  are also called *constant symbols* and are usually denoted by  $a, b, c, d, a_0, c_0, b'$ , etc.

### 3.1 The Alphabet of First-Order Logic

Just as propositional logic formulae, the formulae in first-order logic are strings of characters over a certain alphabet. Unlike propositional logic, the alphabet is now richer. The alphabet of first-order logic consists of the follows “characters”:

1. the logical connectives already known:  $\wedge, \vee, \neg, \rightarrow, \leftrightarrow, \perp$ , as well as two new *quantifiers*:  $\forall, \exists$ ;
2. variables: we will assume that a countably infinite set of variables  $\mathcal{X} = \{x, y, z, x', y', x_1, z'', \dots\}$  is also part of the alphabet (not to be confused with propositional variables in propositional logic – they are two fundamentally different notions);
3. auxilliary symbols: “(”, “)”, “.” and respectively “,”;
4. non-logical symbols, that are specific to each signature: the functional symbols in  $\mathcal{F}$  and the predicate symbols in  $\mathcal{P}$ .

### 3.2 Terms

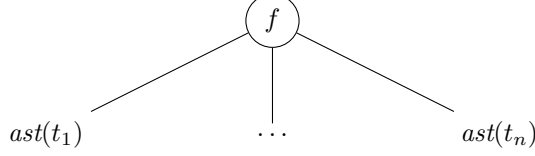
**Definition 3.1.** *The set of termenilor,  $\mathcal{T}$ , is the smallest set having the following properties:*

1.  $\mathcal{F}_0 \subseteq \mathcal{T}$  (any constant symbol is a term);
2.  $\mathcal{X} \subseteq \mathcal{T}$  (any variable is a term);
3. if  $f \in \mathcal{F}_n$  (with  $n > 0$ ) and  $t_1, \dots, t_n \in \mathcal{T}$ , then  $f(t_1, \dots, t_n) \in \mathcal{T}$  (a functional symbol of arity  $n$  applied to  $n$  terms is a term).

We denote terms by the letters  $t, s, t_1, t_2, s_1, t'$ , etc. Even if terms are usually written as a string of characters, they have an associated abstract syntax tree defined as follows:

1. if  $t = c$ ,  $c \in \mathcal{F}_0$ , then  $ast(t) = \bigcirc c$
2. if  $t = x$ ,  $x \in \mathcal{X}$ , then  $ast(t) = \bigcirc x$

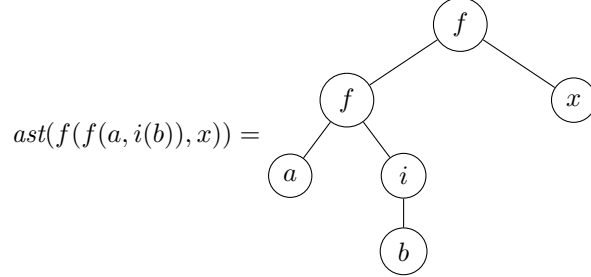
3. if  $t = f(t_1, \dots, t_n)$ ,  $f \in \mathcal{F}_n$  ( $n > 0$ ),  $t_1, \dots, t_n \in \mathcal{T}$ , then  $ast(t) =$



**Example 3.1.** For example, for the signature  $\Sigma = (P, Q, f, i, a, b)$  defined earlier ( $ar(P) = ar(Q) = 2$ ,  $ar(f) = 2$ ,  $ar(i) = 1$ ,  $ar(a) = ar(b) = 0$ ), here are a few examples of terms:  $a$ ,  $b$ ,  $x$ ,  $y$ ,  $x_1$ ,  $y'$ ,  $i(a)$ ,  $i(x)$ ,  $i(i(a))$ ,  $i(i(x))$ ,  $f(a, b)$ ,  $i(f(a, b))$ ,  $f(f(x, a), f(y, y))$ , etc.

Practically, terms are constructed by “applying” functional symbols over constant symbols and variables.

**Example 3.2.** Even if formally terms are defined as strings of characters over the alphabet described above, these must be understood as being trees. In any software program that handles terms, these are stored as a rooted tree. Here is the tree associated to the term  $f(f(a, i(b)), x)$ :



### 3.3 Atomic formulae

**Definition 3.2** (Atomic formula). An atomic formula is any string of characters of the form  $P(t_1, \dots, t_n)$ , where  $P \in \mathcal{P}_n$  is a predicate symbol of arity  $n$ , and  $t_1, \dots, t_n \in \mathcal{T}$  are terms.

**Example 3.3.** Continuing the previous example, we work over the signature  $\Sigma = (P, Q, f, i, a, b)$ , where  $ar(P) = ar(Q) = 2$ ,  $ar(f) = 2$ ,  $ar(i) = 1$ ,  $ar(a) = ar(b) = 0$ .

Here are a few examples of atomic formulae:  $P(a, b)$ ,  $P(x, y)$ ,  $P(f(f(a, i(x)), b), i(x))$ ,  $Q(a, b)$ ,  $Q(i(i(x)), f(x, x))$ , etc.

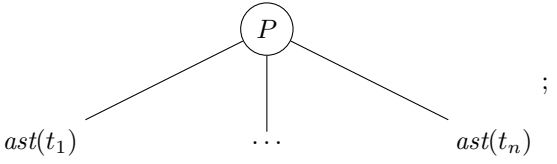
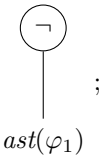
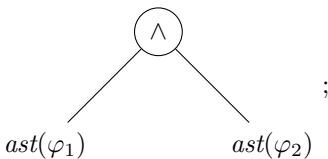
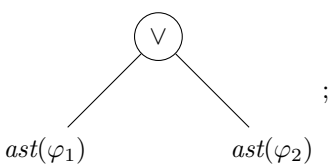
### 3.4 First-Order Formulae

**Definition 3.3** (First-Order Formula). The set of first-order formulae, written *Form*, is the smallest set with the following properties:

1. (base case) any atomic formula is a formula (that is  $P(t_1, \dots, t_n) \in \mathbf{Form}$  for any predicate symbol  $P \in \mathcal{P}_n$  and any terms  $t_1, \dots, t_n$ ; if  $n = 0$ , we write  $P$  instead of  $P()$ );
2. (inductive cases) for any formulae  $\varphi_1, \varphi_2 \in \mathbf{Form}$ , for any variable  $x \in \mathcal{X}$ , we have:
  - (a)  $\neg\varphi_1 \in \mathbf{Form}$ ;
  - (b)  $(\varphi_1 \vee \varphi_2) \in \mathbf{Form}$ ;
  - (c)  $(\varphi_1 \wedge \varphi_2) \in \mathbf{Form}$ ;
  - (d)  $(\varphi_1 \rightarrow \varphi_2) \in \mathbf{Form}$ ;
  - (e)  $(\varphi_1 \leftrightarrow \varphi_2) \in \mathbf{Form}$ ;
  - (f)  $\forall x.\varphi \in \mathbf{Form}$ ;
  - (g)  $\exists x.\varphi \in \mathbf{Form}$ .

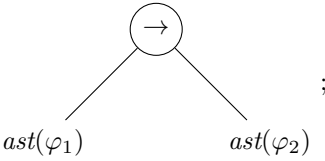
**Remark 3.1.** The predicate symbols of arity 0 play the role of propositional variables (for now, at the syntactic level). The constructions  $\forall x.\varphi$  and  $\exists x.\varphi$  are new.

Formulae have an associated abstract syntax tree defined as follows:

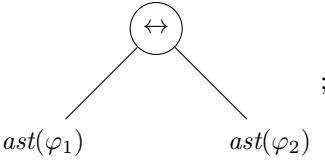
1. if  $\varphi = P(t_1, \dots, t_n)$ , then  $ast(\varphi) =$ 

2. if  $\varphi = \neg\varphi_1$ , then  $ast(\varphi) =$ 

3. if  $\varphi = (\varphi_1 \wedge \varphi_2)$ , then  $ast(\varphi) =$ 

4. if  $\varphi = (\varphi_1 \vee \varphi_2)$ , then  $ast(\varphi) =$ 




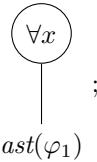
5. if  $\varphi = (\varphi_1 \rightarrow \varphi_2)$ , then  $ast(\varphi) =$



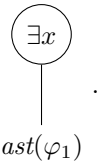
6. if  $\varphi = (\varphi_1 \leftrightarrow \varphi_2)$ , then  $ast(\varphi) =$



7. if  $\varphi = (\forall x.\varphi_1)$ , then  $ast(\varphi) =$



8. if  $\varphi = (\exists x.\varphi_1)$ , then  $ast(\varphi) =$



### 3.5 The Brackets

The brackets, “(” and “)”, are two symbols used to mark the order of carrying out the logical operations (and, or, not, etc.). Next, we will drop certain extra brackets, just like in the case of propositional logic: if a formula can be interpreted as an abstract syntax in two or more ways, we will use brackets to fix the desired tree.

For example, the formula  $\varphi_1 \vee \varphi_2 \wedge \varphi_3$  could be understood as  $((\varphi_1 \vee \varphi_2) \wedge \varphi_3)$  or as  $(\varphi_1 \vee (\varphi_2 \wedge \varphi_3))$ . In order to save brackets, we establish the following priority order of logical connectives:  $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \forall, \exists$ . There is an extra subtleness in dealing with  $\forall$  and  $\exists$  that we discuss in the next lecture. When we are not 100% sure, it is better to use extra brackets.

Because of the order of priority for logical connectives, the formula  $\varphi = \varphi_1 \vee \varphi_2 \wedge \varphi_3$  shall always be understood as  $(\varphi_1 \vee (\varphi_2 \wedge \varphi_3))$  (because  $\wedge$  has priority over  $\vee$ ). As an analogy, it works the same way as in arithmetic:  $1 + 2 * 3$  will be understood as  $1 + (2 * 3)$ , because  $\times$  has priority over  $+$  ( $\times$  is similar to  $\wedge$  and  $+$  to  $\vee$ ).

### 3.6 A First Example

Next, we will explain the signature used to model in first-order logic the statements *any human is mortal*, *Socrates is human* and respectively *Socrates is mortal*.

First, we identify the predicates in the text. We have two unary predicates “is human” and respectively “is mortal”. We choose the predicate symbol  $Human$  for the first predicate and the predicate symbol  $Mortal$  for the second predicate. We also have one constant in the text: Socrates. We choose the arity 0 functional symbol  $s$  for this constant. Therefore, to model the statements above, we shall work in the signature

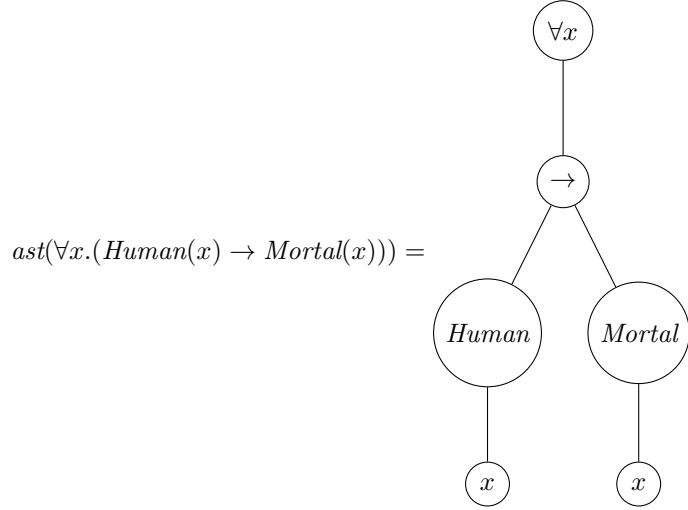
$$\Sigma = (Human, Mortal, s),$$

where  $Human$  and  $Mortal$  are predicate symbols of arity  $ar(Human) = ar(Mortal) = 1$ , and  $s$  is a functional symbol of arity  $ar(s) = 0$ .

The statement *any human is mortal* will be modelled by the first-order formula

$$\forall x.(Human(x) \rightarrow Mortal(x)),$$

whose abstract syntax tree is:



The statement *Socrates is human* shall be modelled by the atomic formula  $Human(s)$ , and the statement *Socrates is mortal* by the atomic formula  $Mortal(s)$ .

For the signature  $\Sigma = (Human, Mortal, s)$  fixed above, there exist several possible  $\Sigma$ -structures. An example of a  $\Sigma$ -structure would be  $S = (D, Human^S, Mortal^S, s^S)$  defined as follows:

1.  $D$  is the set of all beings on Earth;
2.  $Human^S(x)$  is true for any being  $x$  that is human;
3.  $Mortal^S(x)$  is true of any being  $x$  (all of the elements in the domain are mortal);
4.  $s^S$  is Socrates (Socrates, being a being, belongs to the set  $D$ ).

Anticipating a little bit (we shall discuss the semantics of first-order formulae in the next lecture), all tree formulae discussed in this section,  $\forall x.(Human(x) \rightarrow Mortal(x))$ ,  $Human(s)$  and respectively  $Mortal(s)$ , are true in the structure  $S$  defined above.

In fact, the quality of the argument *any human is mortal; Socrates is human; so: Socrates is mortal* is given by the fact that the formula  $Mortal(s)$  is necessarily true in *any* structure in which the formulae  $Mortal(s) \forall x.(Human(x) \rightarrow Mortal(x))$  and respectively  $Human(s)$  are true, not just in the structure  $S$  above.

### 3.7 A Second Example

Consider the signature  $\Sigma = (<, =, +, -, 0, 1)$ , where  $<$  and  $=$  are predicate symbols of arity 2,  $+$  is a functional symbol of arity 2,  $-$  is a functional symbol of arity 1, and 0 and 1 are constant symbols.

Here are a few first-order formulae in the first-order language associated to the signature  $\Sigma$ :

1.  $\forall x.\forall y.(\langle x, y \rangle \rightarrow \exists z.(\langle x, z \rangle \wedge \langle z, y \rangle))$ ;
2.  $\forall x.\forall y.\exists z.(+(x, y), z)$ ;
3.  $\forall x.(\langle 0, x \rangle \vee \langle 0, x \rangle)$ ;
4.  $\forall x.\exists y.(= (x, -(y)))$ ;
5.  $= (+ (x, y), z)$ .

Many times, in the case of binary predicate symbols and binary functional symbols, we use the infix notation (e.g.,  $x < y$  instead of  $\langle x, y \rangle$ ). In this case, we could write the formulae above as follows:

1.  $\forall x.\forall y.(x < y \rightarrow \exists z.(x < z \wedge z < y))$ ;
2.  $\forall x.\forall y.\exists z.(x + y = z)$ ;
3.  $\forall x.(0 < x \vee 0 = x)$ ;
4.  $\forall x.\exists y.(x = -(y))$ ;
5.  $x + y = z$ .

Two of the possible  $\Sigma$ -structures are  $S_1 = (\mathbb{R}, <, =, +, -, 0, 1)$  and  $S_2 = (\mathbb{Z}, <, =, +, -, 0, 1)$ , where the predicates and functions are those known from mathematics (with the remark that  $-$  is the unary minus function).

Anticipating the next lecture, on the semantics of first-order formulae, the first formula is false in  $S_2$  and true in  $S_1$ . The second and the fourth formula

are true both in  $S_1$  and in  $S_2$ . The third formulae is false both in  $S_1$  and in  $S_2$ . The truth value of the fifth formula depends not only of the structure where we evaluate the truth value of the formula, but also on the values of the variables  $x, y, z$ . Because the variables  $x, y, z$  are not protected by a quantifier in formula number 5, they are called *free variables*. Formula number 5 is *satisfiable* both in the structure  $S_1$  as well as in the structure  $S_2$ , because in both cases there are values for the variables  $x, y, z$  that make the formula true (e.g. the values 1, 2, 3 for  $x, y$  and respectively  $z$ ).