Logic(s) for Computer Science
The Syntax of First-Order Logic

November 27, 2018

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 $\mathtt{p} \in A$.

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

No, because to the statement *Socrates is mortal* we should associate a third propositional variable $\mathtt{r} \in A$. We cannot draw any conclusion on $\tau(\mathtt{r})$ from $\tau(\mathtt{p}) = 1$ and $\tau(\mathtt{q}) = 1$. So, from the semantics of propositional logic, we cannot draw the conclusion that $\mathtt{r}$ is true in any truth assignment that makes both $\mathtt{p}$ and $\mathtt{q}$ 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 *Socratse 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 Sets. Relations. Functions.

In this section, we recall some notions we will use in the following sections.

Sets. The concept of *set* probably is the most important and used in mathematics. Formally, the notion of set was introduced as being a collection of objects *well determined and distinct* in which each object appears only once. These objects forming the set are also called *elements*.

There are several ways of representing a set. One way is to enumerate its elements. For instance, if the objects we are working with are the digits, then $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ would represent the set of digits. Shorter, we also can use the notation $\{0, ..., 9\}$ for the set of digits. The other way of representing a set is to use properties that characterize the set's elements. A general form of this type of representation is

$$\{x \mid x \text{ has property P}\}.$$

For instance, the set of digits may also be represented as

$$\{n \mid 0 \leq n < 10, n \text{ is a natural number}\}.$$

Sets are usually noted with capital letters (for instance, A, X, etc.), or eventually with special letters for the known letters ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, etc.). In general, the elements of sets are noted with small letters (for instance, a, x, n, z etc.). To express the fact that an element x belongs to a set X, we use the notation $x \in X$. For instance, we can say that $9 \in \{n \mid 0 \leq n < 10\}$. When an element x *does not* belong to a set A, we use the notation $x \notin A$. For instance, $10 \notin \{n \mid 0 \leq n < 10\}$. We say that a set A is a *subset* of another set B, and we note $A \subseteq B$, if for any element $a \in A$, we have $a \in B$.

Sets may have elements that are sets themselves : $\{\{0\}, \{1\}, \{2\}, \{0, 1\}\}$. Another example of such set is the set of subsets of a set. For some set X, the set of subsets of X is noted with 2^X and contains as elements all subsets of X.

The most common operations over sets are *union*($\cup$), *intersection*($\cap$), *difference*($\setminus$), *symmetric difference*(Δ), *cartesian product* ($\times$), and so on. The cartesian product of two sets A and B is defined as follows:

$$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\},$$

where (a, b) is a ordered pair of elements.

Example 2.1. *For two sets* $A = \{1, 2, 3\}$ *and* $B = \{\alpha, \beta\}$*, the cartesian product is:* $A \times B = \{(1, \alpha), (1, \beta), (2, \alpha), (2, \beta), (3, \alpha), (3, \beta)\}$.

Binary relations. Let A and B two nonempty sets. by definition, a *binary relation* R from A to B is a subset of the cartesian product $A \times B$, that is, $R \subseteq A \times B$. If for an element $(a, b) \in A \times B$ we have $(a, b) \in R$ then we say that a *is in relation* R *with* b and write aRb.

Example 2.2. *A very common binary relation is the relation "less or equal":* $\leq \subseteq \mathbb{N} \times \mathbb{N}$*. The pair* $(2, 3) \in \leq$ *while the pair* $(3, 2) \notin \leq$*. Usually, we use the infixed notation, that is* $2 \leq 3$*, and for* $(3, 2) \notin \leq$ *we write* $3 \not\leq 2$.

For a relation $R \subseteq A \times B$ we call the *domain* of relation R, the set:

$$\{a \in A \mid \text{exists } b \in B \text{ such that } (a, b) \in A \times B\}$$

and the *range* of R the set:

$$\{b \in B \mid \text{exists } a \in A \text{ such that } (a, b) \in A \times B\}.$$

Example 2.3. *For the set* $A = \{1, 2, 3\}$ *and* $B = \{\alpha, \beta\}$*, the set* $R_1 = \{(1, \alpha), (1, \beta), (2, \alpha), (3, \alpha)\}$ *is a binary relation. The domain of* R_1 *is* $\{1, 2, 3\}$ *and its range is* $\{\alpha, \beta\}$.

Functions. *Functions* are particular cases of relations. A binary relation $F \subseteq A \times B$ is a *function* if it fulfills the following two properties:

- the domain of F is the set A;

- for any $a \in A$ and any $b, b' \in B$ such that $(a, b) \in F$ and $(a, b') \in F$ we have $b = b'$.

The first property expresses the fact that for any element a from A, we have some element in B with which a is in relation F. The second property ensures that for any element a from A we have only one element in B with which a is in the relation F.

Example 2.4. *Let $R_1 = \{(1, \alpha), (1, \beta), (2, \alpha), (3, \alpha)\}$ be a relation over the sets $A = \{1, 2, 3\}$ and $B = \{\alpha, \beta\}$. The relation R_1 is not a function because the second property is not fulfilled: to the element $1 \in A$ are associated two distinct elements α and β from B.*

Neither the relation $R_2 = \{(1, \alpha), (2, \alpha)\}$ is not a function because its domain is not A, but only a subset of A.

However, the relation $F_1 = \{(1, \alpha), (2, \alpha), (3, \beta)\}$ is a function. Both properties are fulfilled. Note that: even if for 1 and 2 we associate the same element α, it does not mean that the second property is not respected!

In general, if a binary relation F is a function, instead of $(a, b) \in F$ we write $F(a) = b$.

Cartesian product over n sets. The arity of relations and functions. The definition of cartesian product from above can be generalized for more then two sets. Let take n sets noted $A_1, A_2, \ldots, A_n$. We can define their cartesian product as:

$$A_1 \times A_2 \times \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) \mid a_i \in A_i, i \in \{0, \ldots, n\}\},$$

where $(a_1, a_2, \ldots, a_n)$ is called *n-tuple* or for short *tuple*. By *n-tuple* we understand an ordered sequence of n elements.

If the sets $A_1, A_2, \ldots, A_n$ are equal, that is $A_1 = A_2 = \cdots = A_n = A$, we note by A^n the cartesian product:

$$\underbrace{A \times \cdots \times A}_{n \text{ times}}.$$

Since we defined the cartesian product over several sets, we can naturally generalize the notion of binary relation. Therefore, a relation (not necessary binary) over the sets A_1, A_2, $\cdots$, A_n is defined as a subset of the cartesian product $A_1 \times A_2 \times \cdots \times A_n$. In this case we say that the relation's *arity* is n because each element of this relation is a *n-tuple*. For the particular case when a relation is defined as a subset of A^n we always know that the relation's arity is n. The relation $\leq$ from Example 2.2 is a subset of the cartesian product $\mathbb{N}^2$ and has arity 2.

Example 2.5. *Let take the sets $A = \{1, 2, 3\}$, $B = \{\alpha, \beta\}$ and $C = \{true, false\}$. The set $R = \{(1, \alpha, true), (1, \beta, true), (1, \alpha, false), (1, \beta, false), (2, \alpha, true), (2, \beta, true), (3, \alpha, false), (3, \beta, false)\}$ is a relation over the sets A, B and C. This relation is called* ternary *relation because each element of the relation is a*

sequence with 3 elements - triplets. More generally, the relations whose elements are sequences of n elements, are also called n-ary.

The generalization of relations allow us to define functions with several arguments. Therefore, a relation F over the set A_1, A_2, ..., A_n is a function if:

- the domain of F is $A_1 \times A_2 \times \cdots \times A_{n-1}$ and
- each element $(a_1, a_2, \ldots, a_{n-1})$ from the domain, F associates only one element from A_n.

For functions, instead of $(a_1, \ldots, a_n) \in F$ we use the notation $F\big((a_1, \ldots, a_{n-1})\big) = a_n$. In this case, we say that F has *arity* $n-1$ because it has $n-1$ parameters. However, if we see it as a relation, F has arity n.

Example 2.6. *Let take the sets A, B, C and relation R from Example 2.5. The relation R is not a function. Even if the domain of R is exactly the set $A \times B$, for the element $(1, \alpha) \in A \times B$ the relation R associates both true and false from C.*

On the other hand, the relation $F' = \{(1, \alpha, true), (1, \beta, true), (2, \alpha, true), (2, \beta, true), (3, \alpha, false), (3, \beta, false)\}$ is a function.

When the domain of some function is the empty set, it always returns the same value. These functions are also called *constant* functions or for short *constants*.

Observations

1. Since functions are relations, and the relations are sets, in this section we noted the functions with capital letters (ex. F, F'). In the following sections we will note the functions with small letters as in mathematics: f, g, h, f', f_1 , f_n, etc.
2. In the following sections, we use relations to model the notion of *predicate*.

3 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:

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

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 infixed notation $(x < y)$ instead of the prefixed notation $(<(x, y))$ for many binary predicates (including for $<$).

If the formula φ abnove 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 3.1. *A* mathematical structure *is a tuple* $S = (D, Pred, Fun)$ *where:*

- D *is a non-empty set called the* domain *of the structure;*
- *each* $P^D \in Pred$ *is a predicate (of a certain arity) over the set* D*;*
- *each* $f^D \in Pred$ *is a function (of a certain arity) over the set* D*.*

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

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. $(B, \emptyset, \{\cdot, +, \text{-}\})$;

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

   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:

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

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 $<\in \mathbb{R}^2$ is different from the predicate $<\in \mathbb{Z}^2$, 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 3.2. *A* signature Σ *is a tuple* $\Sigma = (\mathcal{P}, \mathcal{F})$*, where* $\mathcal{P}$ *is a set of predicate symbols and* $\mathcal{F}$ *is a set of functional symbols. Each predicate or functional symbol* s *has an associate natural number called its* arity *denoted by* $ar(s)$*.*

To a signature we can associate many structures:

Definition 3.3. *If* $\Sigma = (\mathcal{P}, \mathcal{F})$ *is a signature, a* Σ-structure *is any structure* $S = (D, Pred, Fun)$ *so that for each predicate symbol* $P \in \mathcal{P}$*, exists a predicate* $P^S \in Pred$ *of corresponding arity, and for every functional symbol* $f \in \mathcal{F}$*, there is a function* $f^S \in Fun$ *of corresponding arity.*

Example 3.1. *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 arrities:* $ar(f) = 2$*,* $ar(i) = 1$ *and* $ar(a) = ar(b) = 0$*.*

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

To remember!
Structure = domain + predicates + functions Signature = predicate symbols + functional symbols To a signature Σ we can associate many structures, which are called Σ-structures.

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 $\mathcal{F}_n = \{f \mid ar(f) = n\}$.

4 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 Σ. In propositional logic, there was just one language, *PL*.

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

4.1 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: $\wedge, \vee, \neg, \rightarrow, \leftrightarrow, \bot$, as well as two new *quantifiers*: $\forall$, $\exists$;
2. variables: we will assume that a countably infinite set of variables $\mathcal{X} = \{x, y, z, x', y', x_1, z'', \ldots\}$ 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 $\mathcal{P}$.

4.2 Terms

Definition 4.1. *The set of* terms, $\mathcal{T}$, *is the smallest set having the following properties:*

1. $\mathcal{F}_0 \subseteq \mathcal{T}$ *(any constant symbol is a term);*

2. $X \subseteq \mathcal{T}$ *(any variable is a term);*

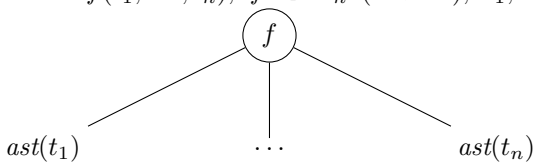
3. *if* $f \in \mathcal{F}_n$ *(with* $n > 0$*) and* $t_1, \ldots, t_n \in \mathcal{T}$*, then* $f(t_1, \ldots, t_n) \in \mathcal{T}$ *(a functional symbol of arity* n *applied to* n *terms is a term).*

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 $ast(t) =$ 

2. if $t = x$, $x \in \mathcal{X}$, then $ast(t) =$ 

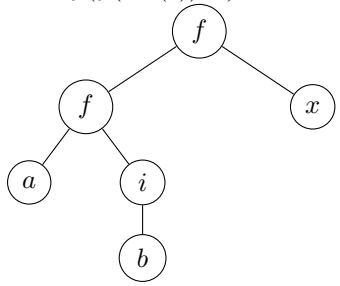
3. if $t = f(t_1, \ldots, t_n)$, $f \in \mathcal{F}_n$ $(n > 0)$, $t_1, \ldots, t_n \in \mathcal{T}$, then $ast(t) =$



Example 4.1. *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.

Remark 4.1. *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* $f(f(a, i(b)), x)$*:*

$ast(f(f(a, i(b)), x)) =$ 

4.3 Atomic formulae

Definition 4.2 (Atomic formula)**.** *An* atomic formula *is any string of characters of the form* $P(t_1, \ldots, t_n)$*, where* $P \in \mathcal{P}_n$ *is a predicate symbol of arity* n*, and* $t_1, \ldots, t_n \in \mathcal{T}$ *are terms.*

Example 4.2. *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: $P(a, b)$*,* $P(x, y)$*,* $P\Big(f\big(f(a, i(x)), b\big), i(x)\Big)$*,* $Q(a, b)$*,* $Q\Big(i(i(x)), f(x, x)\Big)$*, etc.*

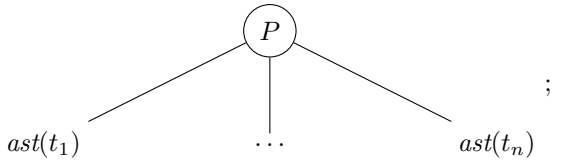
4.4 First-Order Formulae

Definition 4.3 (First-Order Formula)**.** *The set of first-order formulae, written* `Form`*, is the smallest set with the following properites:*

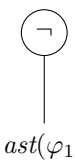
1. *(base case) any atomic formula is a formula (that is* $P(t_1, \ldots, t_n) \in$ `Form` *for any predicate symbol* $P \in \mathcal{P}_n$ *and any terms* $t_1, \ldots, t_n$*; if* $n = 0$*, we write* P *instead of* $P()$*);*
2. *(inductive cases) for any formulae* $\varphi_1, \varphi_2 \in$ `Form`*,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

Remark 4.2. *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.*

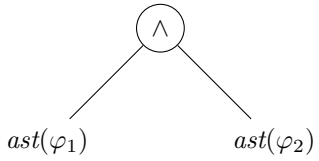
Formulae have an associated abstract syntax tree defined as follows:

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

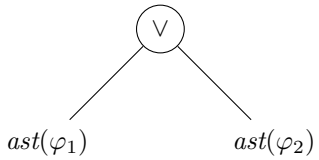
;

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

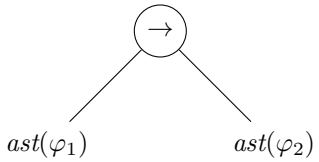
;

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

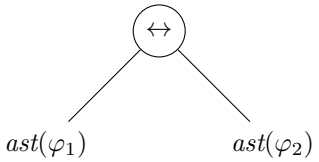
;

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

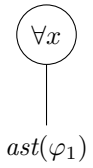
;

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

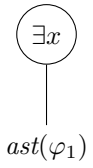
;

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 $ast(\varphi) =$ 

.

4.5 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: $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \forall, \exists$. 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 anology, it works the same way as in arithmetic: $1+2*3$ will be understood as $1 + (2 \times 3)$, because $\times$ has priority over $+$ ($\times$ is similar to $\wedge$ and $+$ to $\vee$).

4.6 A First Example

Next, we will explain the signature used to model in first-order logic the statments *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

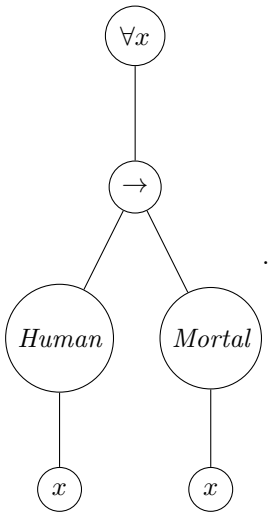
$$\Sigma = (\{Human, Mortal\}, \{s\}),$$

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

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

whose abstract syntax tree is:

$ast(\forall x.(Human(x) \to Mortal(x))) =$ 

.

The statement *Scorates 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 Σ-structures. An example of a Σ-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).

Anticipanting 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.

4.7 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 Σ:

1. $\forall x.\forall y.\Big(< (x, y) \rightarrow \exists z.(< (x, z) \wedge < (z, y))\Big)$;
2. $\forall x.\forall y.\exists z.\Big(= (+(x, y), z)\Big)$;
3. $\forall x.(< (0, x) \vee = (0, x))$;
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 infixed notation (e.g., $x < y$ instead of $< (x, y)$). In this case, we could write the formulae above as follows:

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

4. $\forall x.\exists y.(x = -(y))$;

5. $x + y = z$.

Two of the possible Σ-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).

Anticipanding 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 evalute 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).