1 Syntax for Predicate Logic

Remark 20.1.1 In contrast to propositional logic, first-order logic is best understood as a scheme for languages, depending on the application we have in mind. Each language has a fixed logical component, whose meaning never changes and adds a nonlogical component appropriate to the *domain* under discussion. The domain of discourse is the subject matter and a first-order language is used to make statements about this domain.

We will use lower-case letters at the end of the alphabet x, y, z (possibly with subscripts) as variables standing for arbitrary objects. The domain under discussion is called the *range* of the variables. A first-order language may also add *constant symbols* which are intended as names for specific individuals in the domain. For example, if we are doing number theory our domain is the natural numbers, and there would also be constant symbols naming individual numbers, $0, 1, 2, 3, \ldots$.

We will use capital letters to stand for *properties* of individuals; these letters are called 1-place *predicates*. We will also have 2-place predicates standing for binary relations between individuals, and 3-place predicates standing for ternary relations between individuals. More generally, there may be n-place predicates standing for n-place relations between individuals.

For example, if we are doing number theory, we might have 1-place predicates for property "x is a prime", 2-place predicates for binary relations such as "$x < y$", and 3-place predicates for ternary relations such as "x+y=z". (The relation "$x + y = z$" is the *graph* of the addition function.) Alternatively, our domain of discourse might be the people of the island of knights and knaves, so that x, y, z range over the individuals of the island. So, the language might include names for particular individuals Grogg and Phogg, 1-place predicates for properties like "x is a knight", 2-place predicates for binary relations like "x likes y" and 3-place predicates for ternary relations like "x introduced y to z".

Once we identify the nonlogical component of a language, there will be a uniform syntax for building *formulae* and *statements* (which are the first-order counterpart to propositions), and then a uniform semantics for defining truth for these languages.

The language of first-order logic adds predicates, relations, variables and quantifiers over and above the logical connectives we have seen.

Definition 20.1.2 (syntax of predicate languages) A language $\mathcal{L}$ consists of the following primitive symbols.

The fixed *logical* component of $\mathcal{L}$ consists of the following.

(1) The *logical connectives* are the following symbols we met in propositional logic.

- (a) $\neg$ (*negation*) [read as "not"],
- (b) $\wedge$ (*conjunction*) [read as "and"],
- (c) $\vee$ (*disjunction*) [read as "or"],
- (d) $\rightarrow$ (*conditional*) [read as "if-then"],
- (e) $\leftrightarrow$ (*biconditional*) [read as "if and only if"].
- (f) $\top$ (*universal truth*)
- (g) $\top$ (*universal falsehood*)

This part is the legacy of propositional logic.

(2) There are two *quantifier symbols*:

(h) $\exists$ (*existential quantifier*) [read as "there exists"],

(i) $\forall$ (*universal quantifier*) [read as "for all"].

(3) A countable set $\mathbf{V}$ of *variables*: $x, y, z, x_0, x_1, \ldots, y_0, y_1, \ldots$.

(4) The auxillary symbols are the parentheses $(,)$. (These are introduced to ensure expressions are unambiguous.)

The *nonlogical* component of $\mathcal{L}$, which depends on the language, consisting of:

(5) *constant symbols* **CS**: $c, d, c_0, d_0, \ldots,$ (some set of them, possibly empty, and all of arity 0.)

(6) *Predicate symbols* **PS**: $P, Q, R, P_0, P_1, \ldots$ (there must be at least one predicate symbol). Each predicate symbol has a positive integer associated with it, called its *arity*, which determines the number of places the predicate symbol requires.

We will sometimes use the notation $\mathcal{L}(\mathbf{CS}, \mathbf{PS})$ to denote a first-order language determined by the nonlogical components **CS** and **PS**. We will more usually use the abbreviated form $\mathcal{L}$.

Definition 20.1.3 (terms) A *term* of $\mathcal{L}(\mathbf{CS}, \mathbf{PS})$ is a variable or contant from **CS**. A *ground term* is simply a constant symbol, so will have a definite reference. There need not be any ground terms.

Remark 20.1.4 The formulae of first-order logic are defined inductively. These correspond to propositions in propositional object, they are the statements we make in the language. The base case of the inductive definition is slightly more complicated than that of propositional logic, since we replacing propositional symbols by predicates expressing properties and relations.

Definition 20.1.5 (atomic formulae) For each n-ary predicate symbol P and terms $t_1, \ldots, t_n$, $P(t_1, \ldots, t_n)$ is an *atomic formulae*; also $\top$ and $\bot$ are taken to be atomic formulae.

Remark 20.1.6 The use of commas and parentheses in writing $R(x, y)$ is clearly unnecessary, since there is no difficulty parsing Rxy. All terms consist of a single symbol. The main reason for writing $R(x, y)$ is to make it clear that R is a 2-place connective without having to state it.

Definition 20.1.7 (Formulae) The *formulae* of a first-order language $\mathcal{L}(\mathbf{CS}, \mathbf{PS})$ are defined inductively as follows. The set of first-order formulae are the smallest set of expressions meeting the following conditions:

1. Every atomic formula is a formula.
2. If α and β are formulae then so are each of $(\neg\, \alpha)$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \to \beta)$, $(\alpha \leftrightarrow \beta)$.
3. If x is a variable and α a formula, then each of $((\exists x)\alpha)$ and $((\forall x)\alpha)$ are formulae.

The formula $(\forall x)\alpha$ is called a *universal quantification* with respect to the variable x. The formula $(\forall x)\alpha$ is called an *existential quantification* with respect to the variable x.

Example 20.1.8 If R is a 2-place predicate then the following are formulae:

$$((\forall x)((\forall y)(R(x, y) \to R(y, x))))$$
$$((\forall x)((\forall y)(R(x, y) \to ((\exists z)(R(x, z) \wedge R(z, y))))))$$

It is common in mathematics to write a 2-place predicate using infix notation. For example, we commonly write $x < y$ instead of the prefix form $<(x, y)$. More generally, it is common to write xRy instead of $R(x, y)$. We will adopt this convention when it improves readability and write $x < y$, which should be understood as an informal substitution for the real expression $x < y$. In this case, the formulae above could be written

$$((\forall x)((\forall y)(xRy \to yRx)))$$
$$((\forall x)((\forall y)(xRy \to ((\exists z)(xRz \wedge zRy)))))$$

Definition 20.1.9 (Production Rules) The inductive definition can also be described using production rules, where $form$ is a variable over formulae, $term$ is a variable over terms, and var is a variable over variables.

$$
\begin{array}{lrcl}
(1) & form & \Rightarrow & ((\exists\ var)\ form) \mid ((\forall\ var)\ form) \\
(2) & form & \Rightarrow & (form \wedge form) \mid (form \vee form) \mid (form \rightarrow form) \mid (form \leftrightarrow form) \\
(3) & form & \Rightarrow & (\neg\, form) \\
(4) & form & \Rightarrow & R(term_1, \ldots, term_n) \qquad \text{for each n-ary} R \in \mathbf{PS} \\
(5) & term & \Rightarrow & \mathbf{V} \mid \mathbf{CS} \\
(6) & var & \Rightarrow & \mathbf{V}
\end{array}
$$

Note that there is one production rule (4) for each predicate in **PS**.

Definition 20.1.10 (formation tree for formulae) There is a natural correspondence between our expressions for formulae and formation trees, since they share the same inductive definition. Recall the tree constructors $mknode$ and $mkleaf$

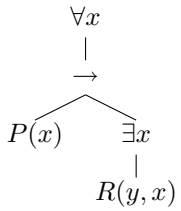
- $mkleaf(s)$ creates a one-node tree with root labeled s.
- $mknode(s, \tau_1)$ creates a tree with root labeled s and one branch τ_1
- $mknode(s, \tau_1, \tau_2)$ creates a tree with root labeled s and and a left branch τ_1 and right branch τ_2

expression	formation tree
$form \Rightarrow R(\ term_1, \ldots, term_n)$	$mkleaf(R(term_1, \ldots, term_n))$
$form \Rightarrow (\neg\, form_1)$	$form = mknode(\neg\,, form_1)$
$form \Rightarrow (form_1 \wedge form_2)$	$form = mknode(\wedge\,, form_1), form_2)$
$form \Rightarrow (form_1 \vee form_2)$	$form = mknode(\vee\,, form_1), form_2)$
$form \Rightarrow (form_1 \rightarrow form_2)$	$form = mknode(\rightarrow\,, form_1), form_2)$
$form \Rightarrow (form_1 \leftrightarrow form_2)$	$form = mknode(\leftrightarrow\,, form_1, form_2)$
$form \Rightarrow ((\exists x)\ form_1)$	$form = mknode(\exists x, form_1)$
$form \Rightarrow ((\forall_x)\ form_1)$	$form = mknode(\forall x, form_1)$

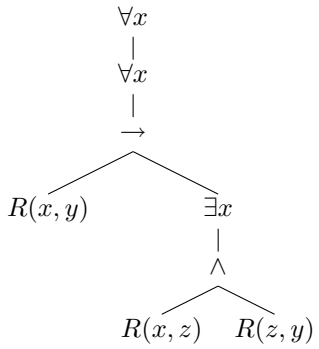
Example 20.1.11 Let P be a 1-place predicate and R be a 2-place predicate. Here is a derivation, using the production rules, showing $((\forall x)(P(x) \rightarrow (\exists y)R(y, x)))$:

$$
\begin{array}{l}
form \overset{(1)}{\Rightarrow} ((\forall\ var)\ form) \overset{(6)}{\Rightarrow} ((\forall x)\ form) \overset{(2)}{\Rightarrow} ((\forall x)(form \rightarrow form)) \\
\quad \overset{(4,5)}{\Rightarrow} ((\forall x)(P(x) \rightarrow form)) \overset{(1,6)}{\Rightarrow} ((\forall x)(P(x) \rightarrow ((\exists x)\ form) \\
\quad \overset{(4,5)}{\Rightarrow} ((\forall x)(P(x) \rightarrow ((\exists x)\ R(y, c))))
\end{array}
$$

The formation tree generated by this derivation is



Example 20.1.12 Here is a formation tree for $((\forall x)((\forall y)(R(x, y) \rightarrow ((\exists z)(R(x, z) \wedge R(z, y))))))$.



Convention 20.1.13 All the conventions for parentheses for propositional logic from Convention 1.2.5 apply as well to first-order formulae. In addition, the quantifiers $\forall$ and $\exists$ will be treated like negation, the have the strongest binding.

For example, we will write the formulae from Example **??** as

$$(\forall x)(\forall y)(xRy \rightarrow yRx)))$$
$$(\forall x)(\forall y)(xRy \rightarrow (\exists z)(xRz \wedge zRy)$$

The strength of binding means that the quantifier in $(\forall x)P(x) \rightarrow Q(x)$ attaches to $P(x)$ as in $((\forall x)P(x)) \rightarrow Q(x)$ and not the conditional as in $(\forall x)(P(x) \rightarrow Q(x))$.