Predicate Logic: Part 1 (Syntax)

CS 4710 Course Notes

February 4, 2019

Predicate Logic

In predicate calculus ("PC"), we defined a single language to model valid, bivalent, truth-functional reasoning. The meaning of statements in PC were taken to be truth values $\top$ and $\bot$ (PC has a *truth-functional semantics*). This was the extent of our ability to refer – to the truth value of a statement.

In the switch to **predicate logic**, we allow some of our vocabulary to refer to objects, functions, and predicates. A statement in PL is meaningful when we can unambiguously determine its truth value. We can do this when of our terms and formulas refer to objects in the right sorts of ways.

We allow our vocabulary to refer to (and predicate properties of) real things as they appear in our day to day experience, whether scientific or mundane. To better serve the purpose of predicating properties of objects (or sets of objects), we must introduce both an expanded syntax and a correspondingly expanded semantics.

Logical Vocabulary

For PC, we defined a single language. In PL, however, we define a *collection* of languages. This is because the *non-logical* vocabulary of PL languages will differ – it's easier to define different languages for different examples than to try and define one massive language for all uses. All PL languages will share logical vocabulary, syntax, and interpretation semantics, which we will see later.

PL's common vocabulary is exactly its logical vocabulary, defined recursively as usual.

Definition. PL's **logical vocabulary** consists of the following categories:
(1) Set of **variables** $\{v_1, v_2, \ldots\}$
(2) Set of **operators** $\{\wedge, \vee, \rightarrow, \neg\}$, inherited from PC
(3) **Quantifiers** $\exists, \forall$
(4) Delimiting pair "(" and ")"

Non-logical Vocabulary

In PL we will name objects in a variety of ways and predicate properties of them. Furthermore, we will be able to construct statements about collections of objects.

Definition. PL language $\mathcal{L}$ has a **non-logical vocabulary**, consisting of the following categories (each of which may be empty, finite, or infinite):

(1) **Predicate symbols** $P_1, P_2, \ldots$

(2) **Names** $n_1, n_2, \ldots$

(3) **Function symbols** $f_1, f_2, \ldots$

(4) Identity symbol "=" (optional), actually a 2-place predicate symbol, but written "infix" (thus "$x = 5$" rather than "$= x5$"). Languages containing the identity symbol are called languages *with identity.*

Each predicate symbol and function has an assigned **arity**, or natural number $n \geq 1$ which represents the number of "inputs" or "places" it has (for instance "Brother(x,y)", meaning "x is y's brother", has arity 2).

A Brief Note on Metalogic vs. Logic

Note that variables such as $\mathcal{L}$ are **metalogical** variables. We use metalogical variables to discuss a system of logic from an external perspective. We also use them to prove things about systems of logic from an external, mathematical perspective. The proofs of soundness and completeness of PC, for instance, are metalogical proofs. They are proofs of mathematical properties of PC, not proofs constructed within PC itself. Note that such a proof would be a derivation (or, in this discussion, a **logical proof**).

Logical Syntax

In PL, our grammatical expressions have a few different types. One type (terms) allow for us to refer to objects themselves. Another type (formulas) allow us to predicate properties of objects and assert relations between them. The final type (statements) are well-formed formulas that permit us to determine a truth value. More on this when we get to semantics.

Definition. $\mathcal{L}$ has a set of **terms**, defined recursively:

(1) Individual names and variables appearing on their own.

(2) If f is a function symbol of arity n (is "n-ary"), and $t_1, \ldots, t_n$ is a sequence of terms (not necessarily distinct), then $f(t_1, \ldots, t_n)$ is a term.

(3) Nothing else is a term.

Definition. $\mathcal{L}$ has a set of **formulas**, defined recursively:

(1) If predicate P is n-ary (has arity n), and $t_1, \ldots, t_n$ are n selections of terms (not necessarily distinct), then $Pt_1 \ldots Pt_n$ is a formula. **(atomic)**

(2) If φ and ψ are formulas, then $(\varphi \vee \psi), (\varphi \wedge \psi), (\varphi \rightarrow \psi), \neg\varphi, \neg\psi$ are all formulas. **(truth-functional)**

(3) If x occurs in φ, and neither $\forall x$ nor $\exists x$ occurs in φ, then $\forall x\varphi$ and $\exists x\varphi$ are both formulas. **(quantified)**

(4) Nothing else is a formula.

Note that formulas can be of many types. For example, $\forall xPx \rightarrow Pa$ is both truth-functional and quantified.

Definition. Suppose we have a formula φ and variable x. Every occurrence of x in φ is either **free** or **bound** (but never both).

(1) If φ is atomic, then all occurrences of x in φ are free.

(2) If φ has the form $\alpha \wedge \beta, \alpha \vee \beta, \alpha \to \beta$ or $\neg\alpha$, then an occurrence of x is free in φ iff that occurrence of x is free in ψ, where ψ is either α or β (depending on which occurrence of x we are considering).

(3) All occurrences of x are bound in $\exists x\varphi$ and $\forall x\varphi$.

If occurrence of x in φ is free, then that occurrence is bound by the occurrence of $\exists x$ or $\forall x$ as in $\exists x\varphi, \forall x\varphi$.

Definition. The set of free variables of φ, written $FV(\varphi)$, is the set of all variables with at least one free occurrence in φ. φ is a **statement** of $\mathcal{L}$ iff $FV(\varphi)$ is empty.

If we assume our $\mathcal{L}$ under consideration has at least one name, we can characterize sentences recursively:

(1) Any n-ary predicate of $\mathcal{L}$ followed by a sequence of n terms. **(atomic)**

(2) For statements P and Q, $P \wedge Q, P \vee Q, P \to Q, \neg P$, and $\neg Q$ are statements.

(3) If P is a statement, and P^* is the statement obtained by swapping all occurrences of a name n in P with variable x (which does not occur in P), then $\exists x P^* x$ and $\forall x P^* x$ are statements.

(4) Nothing else is a statement.

Note types (2) and (3) are unnamed above, and simply referred to as statements.