UNIVERSITY OF CAMBRIDGE

Introductory Logic

Lectures 3 and 4: First-Order Logic

Alan Mycroft

Computer Laboratory, University of Cambridge, UK
http://www.cl.cam.ac.uk/~am21

MPhil in ACS – 2011/12

Lecture Outline

- First-order logic (FOL)
- Syntax
- Semantics, interpretations
- Validity, models, semantic entailment
- Examples
- Deduction, axioms, inference rules
- Soundness, Completeness, the Entscheidungsproblem

First-order logic

This is known by several names, formally “first-order predicate logic” but also “first-order predicate calculus” or, unless qualified otherwise, just “predicate calculus” or “predicate logic”. “FOL” is very common. [Apologies for just calling it ‘predicate logic’ in Lecture 2.]

- It refines the notion of ‘propositional variables’ occurring in wffs into ‘atomic formulae’ which can contain function symbols, relation symbols, and (individual) variables.
- We also add power to talk about ‘for all’ and ‘there exists’ as well as ‘and’, ‘or’, ‘implication’ etc.

NB: propositional variables like A are interpreted as *true* or *false*, while variables in FOL may be interpreted ranging over as arbitrary mathematical objects. The adjective ‘individual’ is used to emphasise the difference when needed.

FOL formally

We assume:

- A countable set $\mathrm{Var} = \{x_1, x_2, \ldots\}$ of *(individual) variables*.
- A countable set $\{F_1, F_2, \ldots\}$ of *function symbols*, each with its *arity*. Constants are just function symbols or arity zero.
- A countable set $\{P_1, P_2, \ldots\}$ of *predicate (or relation) symbols*, each with its *arity*. Old-style "Propositional variables" are just captured as predicate symbols of arity zero. One of these symbols, of arity 2, is distinguished ('picked out') as the equality relation symbol '$=$'.

One logic or many?

What was the purpose of 'countable' above?

- For variables x we really need an unlimited supply. If the logic said there were only 42 variables then there might be some concept we could only express using 43 variables.
- For function symbols and predicate symbols the case is less clear. Historically (and some logicians prefer to have) *one* logic, where any concept which anyone might ever imagine as having an available symbol to represent it.
- The more modern style is to have *one logic* for each situation, and such a logic has a finite number of function and predicate symbols.
- E.g. arithmetic. One possibility: predicate symbols $\{=, <\}$ both of arity 2. and function symbols $\{0, S, +, \times\}$ of arities 0,1,2,2 respectively.

Syntax

Being Computer Scientists and knowing about syntax, trees, bracket-matching etc., we will express the syntax of FOL using BNF. (But remember this is really defining a set of formulae by induction: "the smallest set that . . . ".)
We define (following Enderton):

$$\begin{array}{lrcl} \text{(Terms)} & t & ::= & x \mid F(t_1, \ldots, t_{arity(F)}) \\ \text{Atomic Formulae (Atoms)} & A & ::= & P(t_1, \ldots, t_{arity(P)}) \\ \text{(Wffs)} & \sigma & ::= & A \mid \neg\sigma \mid \sigma \rightarrow \sigma \mid \forall x\ \sigma \end{array}$$

What about '$\exists$', '$\wedge$', '$\vee$', '$\leftrightarrow$' etc? Syntactic sugar:

- we consider $\exists x A$ as shorthand for $\neg\forall x\ (\neg A)$
- since $\neg$ and $\rightarrow$ are universal (Lecture 2) we can consider all other propositional connectives as shorthand.
  E.g. $A \vee B$ represents $(\neg A) \rightarrow B$.

Why ‘first-order’?

- This will become clearer when we give semantics.
- We only allow $\forall$-quantification on values, not on functions or relations.
- Induction using predicate P can be expressed $[P(0) \wedge \forall k\ (P(k) \rightarrow P(S(k)))] \rightarrow \forall n\ P(n)]$, but only by using “one sentence for each predicate”
- The general form: $\forall P\ [P(0) \wedge \forall k\ (P(k) \rightarrow P(S(k)))] \rightarrow \forall n\ P(n)]$ is a statement in higher-order logic, not first-order.
- (An aside) notation: $\forall k\ (P(k))$ can look clumsy, even though it’s quite proper, so some people prefer to define their syntax to use $(\forall k)\sigma$, $\forall k : \sigma$ or $\forall k.\sigma$ (but ‘:’ and ‘.’ often have other meanings in theoretical CS).

Free variables and Sentences

The wff $\forall x\ P(x)$ has no free variables, while the wff $Q(y)$ has y as a free variable. This is a simple *computable* function on syntax – define the set of free variables of a Term or Wff by:

$$
\begin{aligned}
FV(x) &= \{x\} \\
FV(F(t_1,\ldots,t_n)) = FV(P(t_1,\ldots,t_n)) &= \bigcup_{i=1}^{n} FV(t_i) \\
FV(\neg\sigma) &= FV(\sigma) \\
FV(\sigma \rightarrow \sigma') &= FV(\sigma) \cup FV(\sigma') \\
FV(\forall x\ \sigma) &= FV(\sigma) \setminus \{x\}
\end{aligned}
$$

Wffs without free variables are important enough to be given the name *sentences*. Compare: that well-formed *programs* in a language like Java similarly have no free (undeclared) variables.

Semantics

How do we determine when a wff is true, or false? Like the truth assignments in propositional logic, but taking into account the new forms.

An *interpretation* ('*structure*' in Enderton) $\mathcal{I}$ consists of

- a *non-empty* set D (also written $|\mathcal{I}|$) called the *universe of discourse*;
- a function $F^{\mathcal{I}} : D^k \longrightarrow D$ for each function symbol F of arity k.
- a k-ary relation $P^{\mathcal{I}} \subseteq D^k$ for each relation symbol P of arity k. However, we insist that the predicate symbol '=' is always interpreted as equality on D.

This is in principle enough to give a meaning to sentences, but we need to add a (partial) function $v : \mathrm{Var} \longrightarrow D$ giving meaning to free variables (because sentence $\forall x\ \sigma$ contains σ which is in general is only a wff and not a sentence).

Semantics 2

We write $\models_{\mathcal{I},v} \sigma$ to means that $\mathcal{I}, v$ satisfies σ. [There are many variants on this notation: sometimes you see $\mathcal{I}, v \models \sigma$ (conflicts with $\Sigma \models \tau$ elsewhere) and Enderton writes $\models_{\mathcal{I}} \sigma[v]$. The program-semantics notation $[\![\sigma]\!]^{\mathcal{I}}_v$ giving a value in $\mathbb{B}$ is also used.]

First we define the meaning $[\![t]\!]^{\mathcal{I}}_v$ (a value in D) of a term t under $\mathcal{I}, v$:

$$\begin{aligned} [\![x]\!]^{\mathcal{I}}_v &= v(x) \\ [\![F(t_1, \ldots, t_k)]\!]^{\mathcal{I}}_v &= F^{\mathcal{I}}([\![t_1]\!]^{\mathcal{I}}_v, \ldots, [\![t_k]\!]^{\mathcal{I}}_v) \end{aligned}$$

Semantics 3

Now we define $\models_{\mathcal{I},v} \sigma$ on the structure of wffs:

- $\models_{\mathcal{I},v} P(t_1, \ldots, t_k)$ iff $P^{\mathcal{I}}([\![t_1]\!]^{\mathcal{I}}_v, \ldots, [\![t_k]\!]^{\mathcal{I}}_v)$
- $\models_{\mathcal{I},v} \neg\sigma$ iff $\not\models_{\mathcal{I},v} \sigma$
- $\models_{\mathcal{I},v} \sigma \rightarrow \tau$ iff iff $\not\models_{\mathcal{I},v} \sigma$ or $\models_{\mathcal{I},v} \tau$
- $\models_{\mathcal{I},v} \forall x\, \sigma$ iff for every $d \in D$ we have $\models_{\mathcal{I},v[x\mapsto d]} \sigma$

Here $v[x \mapsto d]$ updates the value of a function v at a point x:

$$\begin{aligned} v[x \mapsto d](x) &= d \\ v[x \mapsto d](y) &= v(y) \text{ if } y \text{ and } x \text{ are different variables} \end{aligned}$$

For sentences, note how v merely keeps track of the variables in scope due to '$\forall$' within the recursive definition. So for *sentences* σ, we write $\models_{\mathcal{I}} \sigma$ to abbreviate $\models_{\mathcal{I},\emptyset} \sigma$.

Notes on Semantics

In propositional logic we said, given a wff σ that:

- valuation v *satisfies* σ if $\overline{v}(\sigma) = \mathit{true}$
- σ is *satisfiable* if there is a valuation which satisfies σ
- σ is a *tautology* if every valuation satisfies σ
- σ is *unsatisfiable* if no valuation satisfies σ

Now we say, for each sentence σ:

- σ is *true* in interpretation $\mathcal{I}$ if $\models_{\mathcal{I}} \sigma$. We also say $\mathcal{I}$ is a *model* of σ.
- σ is *satisfiable* if there is an interpretation which makes σ true.
- σ is *valid* if every interpretation makes σ true.

[We're avoiding talking about wffs which are not sentences here.]

Notes

Why don't we use the same words for propositional logic (e.g. 'tautology' meaning true under every truth assignment) and FOL (e.g. 'valid' meaning true in every interpretation)?

- Partly historical accident.
- Partly because it's useful to identify that subset of valid sentences obtained by replacing propositional variables with wffs. In this understanding:
- $\exists x\ P(x) \rightarrow \exists x\ P(x)$ is valid and also a tautology; while
- $\forall x\ P(x) \rightarrow \exists x\ P(x)$ is valid but not a tautology (because it's an instance of $A \rightarrow B$ but not of $A \rightarrow A$).

Semantic entailment

We now define a form of "from these hypotheses we can deduce this consequence" notion. Let Σ be a set of sentences and let τ be a sentence.
Then say:

$\Sigma \models \tau$ *if every model of* Σ *is also a model of* τ*.*

(An interpretation $\mathcal{I}$ is a model of a set of sentences Σ if it is a model of each element.)
I use the phrase "semantic entailment" for $\Sigma \models \tau$; Enderton uses "logically implies" which is used for multiple purposes in the literature, and "tautological implication" when talking about propositional logic. The term "semantic consequence" is also used.

Why do all this?

Semantic entailment, $\Sigma \models \tau$, tells us when one set of sentences require another to be true.

- In propositional logic we could determine *mechanically* whether or not $\Sigma \models \tau$ holds by simply testing every truth assignment.
- In FOL there are an infinite number of interpretations and so we cannot directly determine whether $\Sigma \models \tau$ holds.
- In fact in FOL there is *no* effective means of determining whether $\Sigma \models \tau$ holds.
- However, there is an effective enumeration of all such Σ and τ such that $\Sigma \models \tau$ holds. Why: we'll see later (Gödel's completeness theorem) that all true sentences in FOL have a proof and these proofs can be effectively enumerated.

Some examples

- $\forall x\ \sigma \rightarrow \exists x\ \sigma$ is valid whatever wff σ represents. (We want this to be so, and that's why we require every domain of discourse to be non-empty.)
- $\exists y\ \forall x\ L(x, y) \rightarrow \forall x\ \exists y\ L(x, y)$ is valid.
- $\forall x\ x = x$ is valid.
- $\forall x\ P(x, x)$ is satisfiable, but not valid.
- $\forall x\ \forall y\ x = y$ is true only in interpretations having one element.
- $\exists x\ \exists y\ \exists z\ (x \neq y \wedge x \neq z \wedge y \neq z)$ is true only in interpretations having at least 3 elements. [Formally $x \neq y$ is sugar for $\neg(x = y)$ not a separate predicate symbol.]

Example models

Suppose we have a logic with function symbols $\{0, +\}$ and a set of sentences Σ containing $\forall x \forall y \forall z\ x + (y + z) = (x + y) + z$, $\forall x \forall y\ x + y = y + x$ and $\forall x\ x + 0 = x$. (These are the axioms for a commutative semigroup with identity, but lacking the additive inverse operation a group would have.) Here are 3 models for Σ:

- $(\mathbb{N}; 0, +;)$ is possibly the 'intended' model
- $(\mathbb{R}; 1, \times;)$ is another model, quite expected
- $(\mathbb{B}; \mathit{false}, \vee;)$ is perhaps a surprising model.

Here we're writing *interpretations* as *mathematical structures*: $(D; \mathit{fns}; \mathit{preds})$ where D is a non-empty set supplying the *domain of discourse*, *fns* lists the function interpretations in some agreed order and *preds* lists the predicate interpretations in some agreed order.

Sentences τ such that $\Sigma \models \tau$ are those holding in any such semigroup (this formalises 'theorems which hold in a semigroup')

Deduction

- We've seen how to express *semantic entailment* $\Sigma \models \tau$.
- However, for FOL this is not effectively computable ("for every interpretation $\mathcal{I}$"), even though for propositional logic it *is* computable (there are only 2^n truth assignments for a wff containing n propositional variables).
- So, we want a separate idea of *deducibility* (or *deduction*) written $\Sigma \vdash \tau$. We can think of deduction as "syntactic entailment" – indeed there is a separate course on "automated reasoning (or deduction)" in which the syntactic representation of theorems is of prime importance.
- Ideally we'd like $\Sigma \models \tau$ iff $\Sigma \vdash \tau$ ("validity = provability").
- It turns out this is delicate: Gödel's Completeness Theorem (1930) proves it is the case for FOL, but Gödel's Incompleteness Theorem (1931) says it's impossible when the logic is powerful enough to express arithmetic (which includes higher-order logic).

Inference rules

We need to formalise the concept of *theorem*, *proof* and *inference step within* the logic. Note that we then use the word metatheorem for theorems *about* the logic so we don't get confused.
We need three things:

- a *judgement form* $\mathcal{J}$ giving the syntax of deducible statements, e.g. $\vdash \tau$ or $\Sigma \vdash \tau$ where τ is a wff or a sentence. (We often drop the $\{\ldots\}$ set brackets from the LHS.)
- a set of *axioms*, which are judgements $\mathcal{J}$ expressing things we assume to be true, e.g. $\phi \vdash \phi \vee \psi$ or $\vdash \phi \rightarrow (\phi \vee \psi)$.
  (These axioms effectively say the same thing – in some systems one or the other form will turn out to be more convenient.)
- a set of *inference rules* (proof steps) written $\text{(NAME)}\dfrac{\mathcal{J}_1 \quad \ldots \quad \mathcal{J}_n}{\mathcal{J}_0}$

  (informally: given proofs for $\mathcal{J}_i, 1 \leq i \leq n$ then writing "hence, because (NAME), $\mathcal{J}_0$" after these proofs constitutes a proof of $\mathcal{J}_0$).

Note that an axiom is just a special case ($n = 0$) of a rule.

Proof rules (2)

A *proof* is a tree whose nodes are instances of proof rules and whose leaves are instances of axioms. Trees are typically linearised in English forms of proof by naming judgements and saying (e.g.) "using theorems 1 and 2 and rule 27 we conclude $\langle$whatever$\rangle$".

A judgement is a *theorem* if appears as the judgement at the root ("final line') of a proof.
The rule form enables visual representation of a proof tree:

$$\dfrac{\dfrac{J_0 \quad J_1}{J_2} \qquad \dfrac{J_3 \quad J_4}{J_6}}{J_7}$$

Proof rules (3)

There are two or three main approaches to formalising proof systems for FOL.

Axiomatic (or Hilbert-style) systems These have many axioms and just one or two rules. Enderton gives a system with an infinite number of axioms and just one inference rule, *modus ponens*: MP $\dfrac{\vdash \phi \quad \vdash \phi \rightarrow \psi}{\vdash \psi}$

Natural deduction systems These use just a finite number of rules and axioms, argued to be more representative of how humans reason (they have *introduction and elimination rules* for logical operators. However, the judgement form is of the form $\Gamma \vdash \phi$ and inference rules typically contain judgements containing different Γ components. By contrast rules, like MP above, in axiomatic systems above do not change Γ so this can often be omitted from the formalisation.

Proof rules (4)

Sequent calculi This is another formulation with judgement form $\phi_1, \ldots, \phi_m \vdash \psi_1, \ldots, \psi_n$ which has left- and right-manipulation rules for each logical operator. We'll not discuss it further.

Soundness and Completeness

Given a set R of inference rules (including axioms as the "zero antecedents" special case), we write $\Gamma \vdash_R \phi$ if there is a proof of $\Gamma \vdash \phi$ using the rules in R.
We say a set of rules is

- *sound* if $\Gamma \vdash_R \phi$ implies $\Gamma \models \phi$.
- *complete* if $\Gamma \models \phi$ implies $\Gamma \vdash_R \phi$.

Note that the empty set of rules is sound, but very incomplete while including every sentence as an axiom in R leads to a system which is complete by dint of every sentence being true (but hence very unsound).

The Entscheidungsproblem

We noted earlier that the valid sentences in FOL are recursively enumerable (enumerate all the proofs generated by a sound and complete set of rules).
The question arises to whether we can have a decision (German: Entscheidung) procedure for FOL which, given a sentence or sequent-style judgement, says “yes” or “no” as to whether it is valid or not.
Church and Turing separately proved that no such algorithm can exist.
Turing’s argument is closest to Computer Science:

- Given a Turing machine M, we can exhibit a sentence in FOL which holds if and only if M terminates.
- Since there is no algorithm for this (the halting problem) there can be no decision procedure for validity (or for satisfiablity for FOL).

By contrast this problem is trivially decidable for Propositional Logic.