Notes for Logic (COMP0009)

Raphael Li

Sep 2025

Contents

1 Introduction and revision

Formally, a *logic* consists of three components:

Component	Describes...
Syntax	The language and grammar for writing formulas
Semantics	How formulas are interpreted
Inference system (or proof system)	A syntactic device for proving true statements

Table 1: The three key components of a logic.

This module concerns algorithms that automatically parse and determine the validity of a formula.

1.1 Propositional logic

1.1.1 Syntax

Formulas are constructed by applying negation, conjunction and disjunction to propositions.

$$\begin{aligned} \text{proposition} &\coloneqq p \mid q \mid r \mid \cdots \\ \text{formula} &\coloneqq \text{proposition} \mid \neg\text{formula} \mid (\text{formula} \circ \text{formula}) \qquad \text{(where } \circ \text{ is } \wedge, \vee \text{ or } \rightarrow) \end{aligned}$$

A proposition or its negation is called a *literal*[1].

For any formula that isn't a proposition, the *main connective* is the one with the largest scope. In other words, it is not in the scope of any other connective.

$$((p \wedge q) \vee \neg(q \rightarrow r))$$

This is the connective with which evaluation begins. This is especially important when building parsers for algorithmically evaluating formulas.

1.1.2 Semantics

A valuation is a function v that maps each proposition to a truth value in $\{\top, \bot\}$.

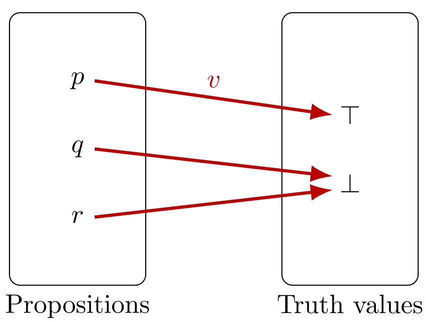

Figure 1: A valuation maps propositions to truth values.

A valuation v can be extended to a unique *truth function* defined on all possible formulas. A truth function v' must satisfy

$$\begin{aligned} v'(\neg\phi) = \top &\iff v'(\phi) = \bot \\ v'(\phi \vee \psi) = \top &\iff v'(\phi) = \top \text{ or } v'(\psi) = \top \\ v'(\phi \wedge \psi) = \top &\iff v'(\phi) = \top \text{ and } v'(\psi) = \top \\ v'(\phi \rightarrow \psi) = \top &\iff v'(\phi) = \bot \text{ or } v'(\psi) = \top \\ v'(\phi \leftrightarrow \psi) = \top &\iff v'(\phi) = v'(\psi) \end{aligned}$$

[1]For example, p and $\neg p$ are both literals, but $\neg\neg q$ is not.

for all formulas ϕ and ψ. From now on we use v to denote the more general truth function.

The result of applying a valuation v to a formula ϕ depends only on the propositional letters that occur in ϕ.

A formula ϕ is *valid* if $v(\phi) = \top$ for all valuations v, which we denote as $\models \phi$. A formula ϕ is *satisfiable* if $v(\phi) = \top$ for at least one valuation v. All valid formulas are satisfiable, but *not* vice versa.

Two formulas ϕ and ψ are *logically equivalent*, written as $\phi \equiv \psi$, if and only if for every valuation v we have $v(\phi) = v(\psi)$.

1.2 First-order logic

1.2.1 Syntax

A first-order language $L(C, F, P)$ is determined by a set C of constant symbols, a set F of function symbols and a non-empty set P of predicate symbols. Each function symbol and predicate symbol has an associated *arity* $n \in \mathbb{N}$. We write f^n and p^n to represent an n-ary function symbol and an n-ary predicate symbol respectively. Moreover, let V be a countably infinite set of variable symbols.

$$\begin{aligned}
\text{term} &\coloneqq c \mid v \mid f^n(\text{term}_0, \text{term}_1, \cdots, \text{term}_{n-1}) && \text{(where } c \in C,\ v \in V \text{ and } f^n \in F\text{)} \\
\text{atom} &\coloneqq p^n(\text{term}_0, \text{term}_1, \cdots, \text{term}_{n-1}) && \text{(where } p^n \in P\text{)} \\
\text{formula} &\coloneqq \text{atom} \mid \neg\text{formula} \mid (\text{formula}_0 \vee \text{formula}_1) \mid \exists v \text{ formula} && \text{(where } v \in V\text{)}
\end{aligned}$$

This definition is functionally complete. Formulas involving universal quantifiers, implications and equivalence symbols can always be rewritten using only symbols defined above.

A *closed term* is a term with no variable symbols. A *sentence* is a formula with no free variables.

1.2.2 Semantics

For a first-order language $L(C, F, P)$, we may construct a corresponding first-order structure[2] $S = (D, I)$ where $I = (I_c, I_f, I_p)$.

$$S = (\underbrace{D}_{\text{non-empty domain}}, \overbrace{(I_c, I_f, I_p)}^{\text{interpretation } I})$$

Here,

- I_c maps each constant symbol in C to an element of D.
- I_f maps each n-ary function symbol in F to an n-ary function over D.
- I_p maps each n-ary predicate symbol $p \in P$ to an n-ary relation over D (i.e. a subset of D^n).
- We may occasionally use I to denote an interpretation function where

$$\begin{aligned}
I(c) &= I_c(c) && \text{(for all } c \in C\text{)} \\
I(f) &= I_f(f) && \text{(for all } f \in F\text{)} \\
I(p) &= I_p(p) && \text{(for all } p \in P\text{)}
\end{aligned}$$

If P includes the equality symbol $=$, then it is always interpreted as the binary relation of true equality.

$$I_p(=) = \{(d, d) : d \in D\}$$

[2] Also known as an L-structure.

Given a structure $S = (D, I)$, a variable assignment A is a map from V to D. For any variable $v \in V$, two variable assignments A and A^* are said to be v-equivalent if $A(x) = A^*(x)$ for all $x \in V \setminus \{v\}$. In other words, two variable assignments are said to be v-equivalent if they are completely identical except possibly for the element in D assigned to v. This is written as $A \equiv_v A^*$.

Given a structure S and a variable assignment A, we may interpret any term as follows.

$$\begin{aligned} c^{S,A} &= I_c(c) \\ v^{S,A} &= A(v) \\ f^n(t_0, t_1, \cdots, t_{n-1})^{S,A} &= \underbrace{(I_f(f^n))}_{\text{interpreted function}}(t_0^{S,A}, t_1^{S,A}, \cdots, t_{n-1}^{S,A}) \end{aligned}$$

Formulas are evaluated as follows.

$$\begin{aligned} S \models_A p^n(t_0, t_1, \cdots, t_{n-1}) &\iff (t_0^{S,A}, t_1^{S,A}, \cdots, t_{n-1}^{S,A}) \in I_p(p^n) \\ S \models_A \neg\text{formula} &\iff S \not\models_A \text{formula} \\ S \models_A (\text{formula}_0 \vee \text{formula}_1) &\iff S \models_A \text{formula}_0 \text{ or } S \models_A \text{formula}_1 \\ S \models_A \exists v \text{ formula} &\iff S \models_{A[x \mapsto d]} \text{formula for some } d \in D \end{aligned}$$

Given a structure S and a formula ϕ, we say that

- ϕ is "valid in S" if $S \models_A \phi$ for every variable assignment A. This is written as $S \models \phi$.
- ϕ is "satisfiable in S" if $S \models_A \phi$ for some variable assignment A.
- ϕ is "valid" if ϕ is valid in all possible structures. This is written as $\models \phi$.
- ϕ is "satisfiable" if there exists some structure in which ϕ is satisfiable.

A formula ϕ is valid if and only if $\neg\phi$ is not satisfiable.

Proof. Let $\neg\phi$ be a formula that is not satisfiable. Hence we have

$$\begin{aligned} \neg\exists S\ \exists A \quad S \models_A \neg\phi &\iff \neg\exists S\ \exists A \quad S \not\models_A \phi \\ &\iff \forall S\ \neg\exists A \quad S \not\models_A \phi \\ &\iff \forall S\ \forall A \quad \neg S \not\models_A \phi \\ &\iff \forall S\ \forall A \quad S \models_A \phi \end{aligned}$$

which means S is valid.

If ϕ is a sentence, then ϕ is valid in S if and only if it is also satisfiable in S.

1.2.3 Example: Arithmetic in the set of natural numbers

Consider the first-order language $L(C, F, P)$ defined as follows. Also assume a countably infinite set V of variable symbols.

$$\begin{aligned} C &= 1, 2, 3, \cdots && \text{(constant symbols)} \\ F &= \{+, \times\} && \text{(function symbols, both binary)} \\ P &= \{=, <\} && \text{(predicate symbols, both binary)} \\ V &= \{x, y, z, \cdots\} && \text{(variable symbols)} \end{aligned}$$

A term is a string of symbols that represents a "thing" or an "object" — this could be a constant, a variable, or a function output.

- x
- $1 + 3$
- $2 \times x + 1$

Of the terms shown above, only the second one is a closed terms because it has no variable symbols.

An atom is a string of symbols that represents the output of a predicate, which is a truth value.

- $1 = 2$
- $y < 3$
- $x + 1 < 2 \times z + 3$

Finally, a formula is constructed by applying negations, disjunctions, and existential quantifiers to atoms.

- $1 = 2 \ \wedge \ y < 3$
- $\neg \exists z \ \ x + 1 < 2 \times z + 3$

The latter example is a sentence because all of its variable symbols are bounded.

For this particular first-order language, we may use the structure of ordinary arithmetic[3], defined as $N = \{\mathbb{N}, \{I_c, I_f, I_p\}\}$ where

- I_c is a function that maps numerical symbols to the corresponding natural number.

$$\begin{gathered} I_c(1) = 1 \\ I_c(2) = 2 \\ I_c(3) = 3 \\ \vdots \end{gathered}$$

- I_f maps $+$ and $\times$ to the addition and multiplication operations in arithmetic respectively.
- I_p maps $=$ and $<$ to the following relations.

$$\begin{gathered} I_p(=) = \{(n, n) : n \in \mathbb{N}\} \\ I_p(<) = \{(m, n) \in \mathbb{N}^2 : m < n\} \end{gathered}$$

1.2.4 First-order structures and directed graphs

Consider a first-order language with only one binary predicate symbol p.

$$L(C, F, \{p\})$$

Any first-order structure $S = \{D, \{I_c, I_f, I_p\}\}$ for this language can be represented as a directed graph, where each vertex is an element of D and each directed edge represents an element of the relation $I_p(p)$.

[3]There is also a similar structure $R = (\mathbb{R}, I)$ where the domain is the set of real numbers.

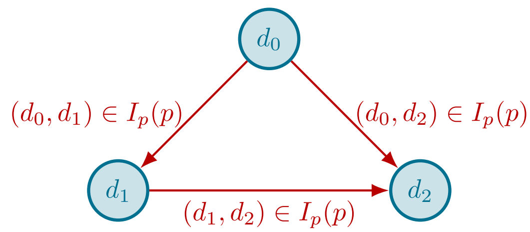

Figure 2: The first-order structure S can be visualised as a directed graph.