Logic Review

- First Order Logic
- Propositional Logic
- Summary

Logic

From Merriam-Webster Online, "Logic is the science of the formal principles of reasoning".

Mathematical logic provides the basis for reasoning the properties and behavior of software systems.

First order logic and propositional logic are two logics that are extensively used in software engineering.
Syntax vs. Semantics

The syntax of a logic is the rules that dictate the composition of legal formulas; the semantics of a logic give precise meaning to each formula. Note that without semantics, syntactic elements are meaningless symbols.

First order logic

First order logic uses variables that range over specific domains such as integers or the reals, relations such as '≤', and functions like '×' or '+'.

It can be used to reason about all the objects in the domain, or to assert that there exists an object satisfying a property.
**Why called first order?**

An important property of this logic is that it only uses simple variables, i.e., those variables that range directly over the predefined domain.

Higher order logics use both simple variables and set variables: Second order logic can use variables that range over sets of objects, third order logic can use variables that range over sets of sets of objects, ...

For instance, let $x$ be a simple variable, and $Y$ a set variable. We can express $x \in Y$ in second order logic but not in first order logic.

---

**Signature**

The signature of a first order logic defines the syntactic objects, i.e., the building blocks to compose a formula.

Formally, $G = (V, F, R)$ includes three sets of disjoint sets: a set of variable symbols $V$, function symbols $F$, and relation symbols $R$. 


**Arity**

The **arity** of a function or relation symbol refers to the number of arguments the symbol can take.

For example, the mathematic function `log` has arity of 1, the mathematic function `add` has arity of 2, and so on.

Note that a **constant symbol** is a function symbol of **arity 0**.

---

**Term**

**Terms** are expressions formed using function symbols and variables. The syntax of a term is formally defined below:

```
term ::= var | const | func (term, term, ..., term)
```

```
add ( add (one, one), v)
```
**Structure**

The *structure* of a first order logic maps *syntactic* objects in a *signature* to *semantic* objects in a *domain*.

Formally, a structure $S = (G, D, F, R, f)$, where $G = (V, F, R)$ is a signature, $D$ is a domain (i.e., a set), $F$ a set of functions, $R$ a set of relations, and a mapping $f : F \cup R \to F \cup R$ from the function and relation symbols in the signature to actual functions and relations over $D$.

---

**Mapping**

A *mapping* must preserve the *arity* of the function and *relation* symbols, which means that a function (or relation) symbol of arity $n$ is mapped to a domain function (or relation) with $n$ parameters.

For instance, the function symbol `sub` can be mapped to the mathematic function `+` over integers.
**Assignment**

An assignment $a$ maps a set $V$ of variables to values in the domain $D$, denoted as $a : V \rightarrow D$.

For instance, let $V$ be the set of variables $\{v_1, v_2, v_3\}$, and $D$ the set of integers. An example assignment is $a = \{v_1 \rightarrow 3, v_2 \rightarrow 0, v_3 \rightarrow -5\}$.

**Term Interpretation**

Let $S = (G, D, F, R, f)$ be a structure. Let $\text{terms}(G)$ be all the terms of a signature $G$. Let $a$ be an assignment.

The term interpretation of $S$ is $Ta : \text{terms}(G) \rightarrow D$, which maps each term in $\text{terms}(G)$ to a value in $D$. $Ta$ can be recursively defined below:

\[
Ta(v) = a(v), \text{ for } v \in V \\
Ta(\text{func}(e_1, e_2, ..., e_n)) = f(\text{func})(Ta(e_1), Ta(e_2), ..., Ta(en))
\]
Example

Let $D$ be the set of integers. Let $a = \{v_1 \rightarrow 2, \, v_2 \rightarrow 3, \, v_3 \rightarrow 4\}$. Let $f$ map add to the add function $+$ over the integers.

\[
\begin{align*}
Ta(v_1) &= a(v_1) = 2 \\
Ta(v_2) &= a(v_2) = 3 \\
Ta(v_3) &= a(v_3) = 4 \\
Ta(\text{add}(v_1, v_2)) &= f(\text{add})(Ta(v_1), Ta(v_2)) = 2 + 3 = 5 \\
Ta(\text{add}(\text{add}(v_1, v_2), v_3)) &= f(\text{add})(Ta(\text{add}(v_1, v_2)), Ta(v_3)) = 5 + 4 = 9
\end{align*}
\]

Simple formula

A simple formula is constructed using relation symbols applied to terms. Formally,

\[
simp\_form ::= \text{rel}(\text{term}, \text{term}, \ldots, \text{term}) \mid \text{term} = \text{term}
\]

For example, $ge (\text{add} (\text{one}, \text{one}), \text{zero})$ is a simple formula.

Important: A term consists of only variables and function symbols, but NOT relation symbols.
First order formula

First order formulas include simple formulas, and can also be formed by applying recursively the Boolean operators and the universal and existential quantifiers. Formally,

\[
\text{form ::= simp\_form | (form \land form) | (form \lor form) | (form \rightarrow form) | (~ form) | (∀ \text{var}(form) | ∃ \text{var}(form) | true | false}
\]

\[
(\geq (\text{one, zero}) \land \geq (\text{add(\text{one, one}), v1}))
\]

\[
\forall \text{v2} (\exists \text{v1} (\geq (\text{v2, v1})))
\]

Precedence

Precedence between Boolean operators can be used to avoid including some of the parentheses. Note that the outermost parentheses can always be ignored.

Usually, \(\neg\) has higher precedence over \(\land\), which in turns has higher precedence over \(\lor\).

For example, \((a \lor (b \land c))\) can be simplified \(a \lor b \land c\).
**Formula Interpretation (1)**

Let $S = (G, D, F, R, A)$ be a structure. Let $\text{forms}(G)$ be all the formulas of a signature $G$. Let $a$ be an assignment.

The formula interpretation of $S$ is $M_a : \text{forms}(G) \rightarrow \{\text{TRUE}, \text{FALSE}\}$, which maps each formula in $\text{forms}(G)$ to a Boolean value.

**Formula Interpretation (2)**

The interpretation $M_a$ of a formula without quantification can be defined below:

1. $M_a(\text{rel}(e_1, ..., e_n)) = f(\text{rel})(Ta(e_1), ..., Ta(en))$
2. $M_a(e_1 = e_2) = (Ta(e_1) = Ta(e_2))$
3. $M_a(f_1 \land f_2)$ is TRUE iff $M_a(f_1) = \text{TRUE}$ and $M_a(f_2) = \text{TRUE}$
4. $M_a(f_1 \lor f_2)$ is TRUE iff $M_a(f_1) = \text{TRUE}$ or $M_a(f_2) = \text{TRUE}$
5. $M_a(f_1 \rightarrow f_2)$ is TRUE iff $M_a(f_1) = \text{FALSE}$ or $M_a(f_2) = \text{TRUE}$
6. $M_a(\neg f_1)$ is TRUE iff $M_a(f_1) = \text{FALSE}$
7. $M_a(\text{true}) = \text{TRUE}$
8. $M_a(\text{false}) = \text{FALSE}$
**Example**

Let \( \delta = \text{ge}(\text{add}(\text{add}(v_1, v_2), v_3), v_2) \wedge \text{ge}(v_3, v_2) \). Let \( a \) be an assignment such that \( a = \{ (v_1, 2), (v_2, 3), (v_3, 4) \} \). Let \( \text{ge} \) is mapped to \( \geq \), and \( \text{add} \) to \( + \).

1. \( M_a(\text{ge}(\text{add}(\text{add}(v_1, v_2), v_3), v_2)) = f(\text{ge})(T_a(\text{add}(\text{add}(v_1, v_2), v_3), T_a(v_2)) = 9 > 3 = \text{TRUE} \)
2. \( M_a(\text{ge}(v_3, v_2)) = f(\text{ge})(T_a(v_3), T_a(v_2)) = 4 \geq 3 = \text{TRUE} \)
3. \( M_a(\delta) = \text{TRUE} \)

**Quantified formulas**

Let \( a \) be an assignment, \( v \) a variable, and \( d \) a value of the chosen domain \( D \). A variant \( a[d/v] \) is an assignment that is the same as \( a \) except that it assigns \( d \) to \( v \). That is, if \( u \neq v \), \( a[d/v](u) = a(u) \); and if \( u = v \), \( a[d/v](u) = d \).

The interpretation of quantified formulas is defined as follows:

- \( M_a(\forall v(\phi)) = \text{TRUE}, \text{ iff for each } d \text{ in } D, M_{a[d/v]}(\phi) = \text{TRUE} \)
- \( M_a(\exists v(\phi)) = \text{TRUE}, \text{ iff there exists } d \text{ in } D \text{ so that } M_{a[d/v]}(\phi) = \text{TRUE} \)
Model, tautology, contradiction

- If \( Ma(\delta) = \text{TRUE} \) under structure \( S \), then we say \( a \) satisfies \( \delta \) under \( S \), denoted as \( a \models^S \delta \).
- If \( a \models^S \delta \) for each assignment \( a \), then \( S \) is a model of \( \delta \).
- If \( \models^S \delta \) for every structure \( S \), then \( \delta \) is a tautology.
- If \( a \models^S \delta \) does not hold for any assignment \( a \) and structure \( S \), then \( \delta \) is a contradiction.

Example

- \( a \models^S x \equiv y \times 2 \), where \( S \) is the structure that includes the domain of integers, and \( \times \) is interpreted as multiplication. This holds if \( a \) assigns 6 to \( x \) and 3 to \( y \).
- \( \models^S x \times 2 \equiv x + x \), where \( S \) includes the domain of integers, and \( x \) and \( + \) are interpreted as usual. Thus, \( S \) is a model of this formula.
- \( \models^S (x \equiv y \land y \equiv z) \rightarrow x \equiv z \) is a tautology.
Syntax vs Semantics Revisited

Consider the following formula:

\[ \varphi = \forall v_1 \forall v_2 (v_1 < v_2 \rightarrow \exists v_3 (v_1 < v_3 \land v_3 < v_2)) \]

- Is this formula TRUE or FALSE?
- Now, assume that \(<\) does represent less than. Is this formula TRUE or FALSE?
- Furthermore, assume that the intended domain is integers, is this formula TRUE or FALSE?
- What if the intended domain is reals?

Logic Review

- First Order Logic
- Propositional Logic
- Summary
**Syntax**

In **propositional logic**, formulas are formed using the following rules:

\[
\text{form ::= prop | (form \land form) | (form \lor form) | (form \rightarrow form) | \neg form | true | false}
\]

where prop is a variable over a set of propositional variables \(\text{AP}\). Each variable in \(\text{AP}\) ranges over the Boolean values \{TRUE, FALSE\}.

**Semantics**

An assignment \(a\) maps a propositional variable in \(\text{AP}\) to a Boolean value. Formally, \(a : \text{AP} \rightarrow \{\text{TRUE, FALSE}\}\).

The interpretation of a formula in propositional logic is defined as in first order logic. That is, \(\text{Ma}(\text{prop}) = a(\text{prop}), \text{Ma}(\text{f}1 \land \text{f}2) = \text{Ma}(\text{f}1) \land \text{Ma}(\text{f}2)\), and so on.

Note that there do not exist the notions of **signature** and **structure**. As a result, we can simply write \(a \models \varphi\) when \(\text{Ma}(\varphi) = \text{TRUE}\).
Tautology and contradiction

A propositional formula is a **tautology** if it is satisfied by any assignment, and is a **contradiction** if there is no assignment satisfying it.

\[ P \lor \neg P \quad P \land \neg P \quad \neg P \land (Q \lor P) \]

Propositional vs First Order

What is the difference between **propositional** and **first order** logic?

Propositional logic does not have quantification, function and relation symbols.
Logic Review

- First Order Logic
- Propositional Logic
- Summary

Summary

- **Logic** provides the basis for reasoning software systems.
- **First order** logic can only use **simple** variables. **Higher order** logics can use **set** variables.
- The $\forall$ quantifier allows to reason properties about all the objects in the domain.
- The $\exists$ quantifier allows to assert properties that are satisfied by at least one object.
- **Propositional logic** is a **simpler** formalism than **first order logic**.