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 ranger over sets of objects, third order logic can use variables that ranger 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 $f : F \cup R \rightarrow F \cup R$ 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 {v1, v2, v3}, and D the set of integers. An example assignment is a = {v1 $\rightarrow$ 3, v2 $\rightarrow$ 0, v3 $\rightarrow$ -5}.

Term Interpretation

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

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

$$Ta(v) = a(v), \text{ for } v \in V$$
$$Ta(func(e1, e2, \ldots, en)) = f(func)(Ta(e1), Ta(e2), \ldots, Ta(en))$$

Example

Let D be the set of integers. Let a = {v1 → 2, v2 → 3, v3 → 4}. Let f map add to the add function + over the integers.

Ta(v1) = a(v1) = 2
Ta(v2) = a(v2) = 3
Ta(v3) = a(v3) = 4
Ta(add(v1, v2)) = f(add)(Ta(v1), Ta(v2)) = 2 + 3 = 5
Ta(add(add(v1, v2), v3)) = f(add)(Ta(add(v1, v2)), Ta(v3)) = 5 + 4 = 9

Simple formula

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

simp_form ::= rel(term, term, …, term) | term ≡ term

For example, ge (add (one, one), 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,

form ::= simp_form | (form $\wedge$ form) | (form $\vee$ form) | (form $\rightarrow$ form) | ($\neg$ form) | ($\forall$var(form) | $\exists$var(form) | true | false

(ge (one, zero) $\wedge$ ge (add(one, one), v1))

$\forall$v2 ($\exists$v1 (ge (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 $\wedge$, which in turns has higher precedence over $\vee$.

For example, (a $\vee$ (b $\wedge$ c)) can be simplified a $\vee$ b $\wedge$ c.

Formula Interpretation (1)

Let $S = (G, D, F, R, f)$ be a structure. Let forms(G) be all the formulas of a signature G. Let a be an assignment.

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

Formula Interpretation (2)

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

1. $Ma(rel(e1, \ldots, en)) = f(rel)(Ta(e1), \ldots, Ta(en))$
2. $Ma(e1 \equiv e2) = (Ta(e1) = Ta(e2))$
3. $Ma(f1 \wedge f2) = TRUE$ iff $(Ma(f1) = TRUE$ and $Ma(f2) = TRUE)$
4. $Ma(f1 \vee f2) = TRUE$ iff $(Ma(f1) = TRUE$ or $Ma(f2) = TRUE)$
5. $Ma(f1 \rightarrow f2) = TRUE$ iff $(Ma(f1) = FALSE$ or $Ma(f2) = TRUE)$
6. $Ma(\neg f1) = TRUE$ iff $(Ma(f1) = FALSE)$
7. $Ma(true) = TRUE$
8. $Ma(false) = FALSE$

Example

Let δ = ge(add(add(v1, v2), v3), v2) $\wedge$ ge(v3, v2). Let a be an assignment such that a = { (v1, 2), (v2,3), (v3, 4)}. Let ge is mapped to $\geq$, and add to +.

1. Ma(ge(add(add(v1, v2), v3), v2)) = f(ge)(Ta(add(add(v1, v2), v3), Ta(v2)) = 9 > 3 = TRUE
2. Ma(ge(v3, v2)) = f(ge)(Ta(v3), Ta(v2)) = 4 $\geq$ 3 = TRUE
3. Ma(δ) = 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:

$M_a(\forall v(\phi))$ = TRUE, iff for each d in D, $M_{a[d/v]}(\phi)$ = TRUE

$M_a(\exists v(\phi))$ = TRUE, iff there exists d in D so that $M_{a[d/v]}(\phi)$ = TRUE

Model, tautology, contradiction

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

Example

- $a \models^s x \equiv y \times 2$, where S is the structure that includes the domain of integers, and × 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.
- $\models (x \equiv y \wedge y \equiv z) \rightarrow x \equiv z$ is a tautology.

Syntax vs Semantics Revisited

Consider the following formula:

$\varphi = \forall v1\ \forall v2\ (v1 < v2 \rightarrow \exists v3\ (v1 < v3 \wedge v3 < v2))$

- 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:

form ::= prop | (form $\wedge$ form) | (form $\vee$ form) | (form $\rightarrow$ form) | $\neg$ form | true | false

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

Semantics

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

The interpretation of a formula in propositional logic is defined as in first order logic. That is, Ma(prop) = a(prop), Ma(f1 $\wedge$ f2) = Ma(f1) $\wedge$ Ma(f2), 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 Ma(φ) = 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.

$P \vee \neg P$ $P \wedge \neg P$ $\neg P \wedge (Q \vee P)$

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.