First Order Logic: Syntax and Semantics

Mikhail Soutchanski

(Acknowledgements. This handout is based on the following materials:
Mordechai Ben-Ari “Mathematical Logic for Computer Science"
Stephen Cook “Predicate Calculus", handout for CSC 438F/2404F)

July 11, 2025

Syntax of first order language

Definition

A first-order *language* (or *vocabulary*, or signature) is specified by

1. a set of n-place *function symbols* that can be empty, where $n \geq 0$ is an integer representing the number of arguments. A zero-argument function is called a constant symbol. We use letters f, g, h possibly with subscripts to denote function symbols.
2. a set of *predicate symbols* such that each predicate symbol has a positive number of arguments (called *arity*). We use letters P, Q, R possibly with subscripts to denote predicate symbols.

To build first-order formulas we also use the following symbols:

- variables that we usually denote as $x, y, z, \ldots$
- propositional connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ (not, and, or, if-then, iff)
- quantifiers $\forall, \exists$ (for all, exists)
- parentheses

In propositional logic, we defined which strings are wffs. We need a similar definition here. Moreover, we have to define terms built by composition of function symbols, variables and constants. For example, in arithmetics, we can compose complex arithmetical expressions out of function symbols $+, \cdot$.

In several textbooks, the symbols starting with upper-case letters are constants, and the symbols starting with lower-case letters are variables.

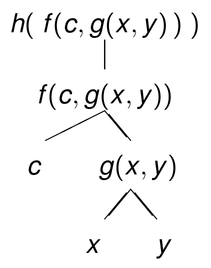
Terms in first order logic (FOL)

Definition (1) Every variable is a term. (2) Every constant is a term.
(3) If f is n-place function symbol in the language, and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.
According to this definition, *terms* (or *expressions*) are strings composed from constants, variables and function symbols. These strings represent objects in the universe of discourse.

Examples: if h is unary function symbol, f, g are binary function symbols, x, y are variables, c is a constant, then all of the following expressions are terms:

$$c, x, y, h(c), h(x), h(y), f(c, c), g(c, c), f(x,y), g(x,y), f(y,x), g(y,x), f(x,c), g(x,c), f(y,c), g(y,c), f(c,x), g(c,x), f(c,y), g(c,y), h(h(c)), h(f(x,y)), h(g(x,y)), \ldots$$

Term $h(f(c, g(x, y)))$ can be represented using a binary tree:



Term formation tree is unique

Definition

A term formation tree associated with the term in the root node is an ordered finite branching tree T labeled with terms satisfying the following conditions:

- The leaves of T are labeled with those variables or constant symbols which occur in the root.
- Each non-leaf node of T is labeled with a term of the form $f(t_1, \ldots, t_n)$.
- A node of T that is labeled with a term of the form $f(t_1, \ldots, t_n)$ has exactly n children in the tree. They are labeled in order with $t_1, \ldots, t_n$.

Example. The vocabulary of formal arithmetic has only one constant **0**, unary function symbol **s** representing successor, binary function symbols $+, \cdot$ and binary predicate symbol $=$. When we write terms in this language in practice we write functions $+, \cdot$ as though they were infix operators, even officially they should be written using prefix notation. In other words, we write $(t_1 \cdot t_2)$ instead of $\cdot(t_1, t_2)$ and we write $(t_1 + t_2)$ instead of $+(t_1, t_2)$.

Exercise: identify all sub-strings which are arithmetical terms in the following arithmetical expression: $(\,(\,((x+y)\cdot(x+s(0)))\cdot(y+s(s(0)))\,)+s(s(s(0))))$. Here $s(x)$ represents a positive integer number following after the number x, e.g., $s(0)$ would represent one, $s(s(0))$ represents two, and so on. We need $s(x)$ because there are no other constants in the language. Draw the term formation tree for this expression.

Motivation for first-order formulas

First order logic (FOL) is expressive enough to show internal structure of simple English sentences.

Roughly speaking, simple FOL terms are the noun phrases of first-order languages: constants can be thought of as first-order counterparts of proper names (such as Bob, Mary), and variables as first-order counterparts of pronouns (his, its, my) or nouns.

We can then combine our ‘noun phrases’ with our various ‘predicate symbols’ to form what we call an atomic FOL formula. Intuitively, an atomic formula is the first-order counterpart of a simple natural language sentence.

Example: the simple sentence “*Bob loves Mary*" can be represented as an atomic formula $loves(Bob, Mary)$ using the binary predicate “*loves*" that has two constants “*Bob*" and “*Mary*" as its arguments.

Definition of first-order formula

A *well-formed FOL formulas* are defined inductively similar to terms:

- Let R be n-place predicate symbol ($n \geq 1$), and $t_1, \ldots, t_n$ are terms, then $R(t_1, \ldots, t_n)$ is an (atomic) well-formed formula. (Note that in the last sentence we need exactly n terms to occupy n argument positions available in R).
- Let t_1, t_2 be well-formed FOL terms, then the equality between these terms $(t_1 \;=\; t_2)$ is also a well-formed formula.
- If F_1, F_2 are well-formed formulas, then so are $(\neg F_1)$, $(F_1 \vee F_2)$, $(F_1 \wedge F_2)$, $(F_1 \rightarrow F_2)$, $(F_1 \leftrightarrow F_2)$.
- If v is a FOL variable and F is a well-formed formula, then $(\exists v)F$ and $(\forall v)F$ are also well-formed formulas.

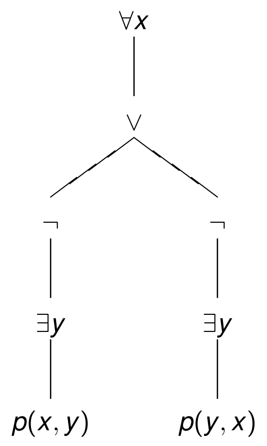
Nothing else can be a *well-formed formula* (wff).

In practice, we often omit parentheses around quantifiers. However, we have to keep parentheses to indicate scopes of quantifiers. Example: a string $\exists x(F_1(x) \vee F_2(x))$ is a well-formed formula, but $\exists x(F_1(x)) \vee F_2(x)$ is not. If in doubt, keep parentheses!

Formula formation tree is unique

Similarly to propositional logic, we can define a formation tree for any first order logic formula. The main difference is that now formulas may involve quantifiers, and each atomic formula is built using terms constructed using terms formation trees as defined above.

A formation tree for $\forall x(\neg\exists y\, p(x, y) \vee \neg\exists y\, p(y, x))$, where parentheses around both occurrences of $\exists y$ in the formula are omitted to avoid clutter:



Free and bound occurrences of variables

In formulas $(\exists v)F$ and $(\forall v)F$, the sub-string F is called the *scope* of the quantifier. Notice that it might happen that F does not contain the variable v.

An occurrence of a FOL variable v is said to be *bound* in a well-formed formula F if either it is the occurrence of v in a quantifier $(\forall v)$, or in a quantifier $(\exists v)$, respectively, or it lies within the scope of a quantifier $(\forall v)$ inside F, of a quantifier $(\exists v)$ inside F, respectively.

Otherwise, a FOL variable that is an argument of one of the predicates or one of the function symbols in F is said to be *free* in F.
Caution: a quanitified variable is assumed to be bound by the nearest quantifier, if it is within the scopes of several nested quantifiers over this var.

Examples: in the formula $\forall x(\neg\exists y\, p(x, y) \vee \neg\exists y\, p(y, x))$ all occurrences of variables x, y are bound. This is displayed in the formation tree of this formula since all quantifiers are above all occurrences of variables in this tree.

Examples: in the formula $\forall x(\neg p(x, y) \vee \neg\forall z\, p(y, x))$ all occurrences of variable x bound, the only one occurrence of variable z is also bound, but all occurrences of variable y are free. Notice there are no occurrences of z within the scope of the quantifier $\forall z$.
Exercise: find all free and bound occurrences of variables in

$$\forall x\big(\exists x(\, x = y \wedge \forall y \forall x(x = y \vee y = z)\,)\big).$$

Examples: translate simple sentences into FOL

This syntax of FOL allows us to represent internal structure of simple English sentences.

For example, the English sentence “all humans are mortal” can be written in FOL as $\forall x(\mathit{human}(x) \rightarrow \mathit{mortal}(x))$ using predicates $\mathit{human}(x), \mathit{mortal}(x)$.

The Greek philosopher Aristotle (384-322B.C.) developed *syllogism*, a form of logical argument. One of his well-known arguments “All humans are mortal. I am a human. Consequently, I am mortal" can be written in FOL using constant I as follows:

$$(\forall x(\mathit{human}(x) \rightarrow \mathit{mortal}(x)) \wedge \mathit{human}(I)) \rightarrow \mathit{mortal}(I).$$

The English sentence “everyone has a mother" can be written in FOL as $\forall p \exists m(\mathit{mother}(p) = m)$ using the function symbol $\mathit{mother}(x)$. Propose FOL-rendering for the sentence “everyone has one and the only one mother".

The sentence “everyone has one and the only one mother" can be written as

$$\forall p \exists m\Big((\mathit{mother}(p) = m) \wedge \forall x(\, \neg(x = m) \rightarrow \neg(x = \mathit{mother}(p))\,)\Big).$$

Web of beliefs in FOL

FOL might be a language of choice to represent a vast web of beliefs in a knowledge base about an application domain. Recall “intelligence as symbol manipulation" hypothesis. We assume that knowledge can be represented symbolically using “shared concepts" between sentences and that collectively they represent meaning of all the concepts.

We can consider as an example a sentence "*Someone is a bachelor if and only if there has never been a wedding where that person is the groom*".

Using 1-place predicate $\mathit{bachelor}(x)$, another 1-place predicate $\mathit{wedding}(y)$, and assuming that y is a wedding event that has an associated participant who is represented using 1-place function $\mathit{groom}(y)$, we can represent this sentence in FOL. How?

$$\forall x\Big(\mathit{bachelor}(x) \leftrightarrow \neg\exists y(\, \mathit{wedding}(y) \wedge (x = \mathit{groom}(y))\,)\Big).$$

Exercise

Consider the following predicates. $H(x)$ means “x is a human". $C(x)$ means “x is a car". $T(x)$ means “x is a truck". $D(x, y)$ means “x drives y". Using these predicates, write formulas representing the following statements.

- “Everybody drives a car or a truck".
  $\forall x(H(x) \rightarrow \exists y(D(x, y) \wedge (C(y) \vee T(y)))\,)$.
- “Some people drive both"
  $\exists x(\, H(x) \wedge \exists y(D(x, y) \wedge C(y)) \wedge \exists z(D(x, z) \wedge T(z))\,)$.
- “Some people do not drive either"
  $\exists x(H(x) \wedge \neg\exists y(D(x, y) \wedge (C(y) \vee T(y))))$.
- “Nobody drives both"
  $\neg\exists x(\, H(x) \wedge \exists y(D(x, y) \wedge C(y)) \wedge \exists z(D(x, z) \wedge T(z))\,)$.

Translate ambiguous sentences into FOL

One of the main strength of FOL comes with its ability to formulate precisely different readings of ambiguous English sentences where ambiguity arises due to uncertainty in the scope of quantifiers.

The well-known example, so-called "donkey phrase", was proposed by Peter Geach (1962): "*Everyone who owns a donkey beats it*".

To represent this sentence we can use binary predicate $\mathit{owns}(p, d)$ meaning “a person p owns a donkey d" and binary predicate $\mathit{beats}(p, d)$ (a person p beats a donkey d). There are several different readings of this sentence.

- “there is donkey (a single poor creature) such that every peasant who owns this donkey beats it".
- “every peasant who has his own donkey beats his donkey".

Propose how these two different readings can be written in FOL.

The first reading: $\exists d \forall p(\mathit{owns}(p, d) \rightarrow \mathit{beats}(p, d))$.

The second possible reading: $\forall p \exists d(\mathit{owns}(p, d) \rightarrow \mathit{beats}(p, d))$.

The barber paradox in FOL

FOL was instrumental to clarify the foundations of mathematics, a discipline where the very basic concept of set turned out to be ambiguous at the end of XIX century.

This might be hinted with a Barber Puzzle, an overly simplified, but popular version of Bertrand Russell's paradox about sets. Russell states it as follows: "You can define the barber as *one who shaves all those, and those only, who do not shave themselves.* The question is does the barber shave himself?"

Formulate this paradox in FOL using 1-place predicate symbol $barber(x)$ and 2-place predicate symbol $shaves(x,y)$ that means "x shaves y".

This puzzle can be written as
$\exists x(\ barber(x) \leftrightarrow \forall p(shaves(x,p) \leftrightarrow \neg shaves(p,p)\,)\,)$
Using the well-defined semantics of FOL, one can prove that negation of this formula holds no matter how we interpret the predicates in the puzzle.

Take home message: intelligent humans such as mathematicians can be wrong when they operate with abstract concepts. If we would like our AI system to reason correctly, its reasoning should be verifiable according to semantics of FOL. More specifically, the fact whether $KB \models Query$ for a given KB and $Query$ should be determined based on semantics of FOL.

Goldbach's conjecture (1742).

"Every even integer number greater than 2 is the sum of two primes" ($1.6 \cdot 10^{18}$)
Formulate this in FOL using unary predicate symbols $Even(x)$ and $Prime(x)$, the binary predicate symbol $>$ (use common infix notation), a binary function symbol $+$ (with usual infix notation), and a constant symbol 2.

$$\forall x(\ (Even(x) \wedge x > 2) \rightarrow \exists y \exists z(Prime(y) \wedge Prime(z) \wedge x = y + z)\ )$$

This can be written as a formula in the language of formal arithmetics since the predicates $Even$, $Prime$, and $>$ can be defined in terms of $s(x), +, \cdot$, and $=$. For example $Even(x)$ can be an abbreviation for the formula $\exists y(x = y + y)$. Exercise: define $Prime(z)$ in the language of arithmetics.

Solution: $Prime(z)$ is an abbreviation for
$$\forall x \forall y(z = x \cdot y \rightarrow (\,(x = s(0) \wedge y = z) \vee (x = z \wedge y = s(0))\,)\,).$$
Exercise: identify which occurrences of variables in this string are bound and which occurrences are free.

In the formula $\exists y(x = y + y)$ the only occurrence of x is free, while all 3 occurrences of y are bound.
Notice that a variable can have both free and bound occurrences in one formula. For example, in $(P(x) \wedge \forall x\, Q(x))$, the first occurrence of x is free, and the second occurrence of x is bound.

Intuitively, the meaning of a formula with free variables depends on the values assigned to its free variables. However, no value need to be assigned to a bound variable to give meaning to the formula.

Semantics of FOL

In propositional logic, a truth assignment interprets a formula. In FOL, we need a more complicated object called a *structure* (or *FOL interpretation*) to give meaning to strings representing quantified formulas and terms.

Let L be a first-order language, then an L-structure $\mathcal{M}$ consists of:

1. A non-empty set D called the *domain*, or the *universe of discourse*. Variables in an L-formula range over D.
2. For each n-ary function symbol f in L, an associated function $f^{\mathcal{M}}$: $D^n \mapsto D$ that maps every n-tuple of elements into a unique element in D.
3. For each n-ary predicate symbol P in L, an associated relation $P^{\mathcal{M}} \subseteq D^n$. If L contains $=$, then $=^{\mathcal{M}}$ must be the usual equality relation on D.

This definition is not about modeling mathematical notions. Its intention is to model everyday reality in precise terms so that we can write programs. DBs.

A formula or a term is called *closed* if it contains no free occurrences of a variable. A closed formula is called a *sentence*.

Every sentence becomes either true or false when interpreted by a structure $\mathcal{M}$ (to be explained soon). If a sentence F becomes true under $\mathcal{M}$, then we say $\mathcal{M}$ *satisfies* F, or $\mathcal{M}$ is a *model* for F, and write $\mathcal{M} \models F$. But for a wff with free object variables, we have to assign values to variables before we can determine whether a formula is true.

Object assignment

We say that a structure (interpretation) $\mathcal{M}$ is *finite* if the universe D of $\mathcal{M}$ is finite. Otherwise, $\mathcal{M}$ is infinite. We always assume universe D is non-empty.

If a formula F has free variables, then these variables must be interpreted as specific elements in the universe D before F gets a truth value under $\mathcal{M}$.

Definition. An *object assignment* σ (read as "sigma") for a structure $\mathcal{M}$ is a mapping from (all) variables to the universe D of $\mathcal{M}$.

Example: the formula $\exists z(\ z < (x_1 + x_2)\,)$ is neither true, nor false over the domain of integers. It all depends on how variables x_1, x_2 are interpreted. Let the universe D be a set of integers≥ 0 with the usual arithmetical operations on them. Under the object assignment $\sigma(x_1){=}2$, $\sigma(x_2){=}5$, $\sigma(x_i){=}0$ for all other vars, this formula is saying there exists a number less than 7, which is true. However, if $\sigma(x_i){=}0$ for all $i \geq 0$, then this formula is false since there is no positive integer that is less than 0.

Notice σ can assign same or distinct values to some or all of the variables. In any case, for each σ, every variable is assigned a single value by σ.

If an object assignment is exactly as σ except that it assigns an element $d \in D$ to variable x, we call it a variant object assignment $\sigma(x \mapsto d)$. By definition, $\sigma(x \mapsto d)(x){=}d$, or in words, $\sigma(x \mapsto d)$ assigns an element d to x.

Object assignment σ: Example

Consider first order language that includes constants such as *Elizabeth*, *Charles*, *William* and so on, two 1-place function symbols $f(x), m(y)$, and the 2-place predicate $=$.

To interpret terms and formulas of this language, structure $\mathcal{M}$ consists of a domain D of people such that each constant represents a person in D. $\mathcal{M}$ interprets $f(x), m(y)$ as the functions that map arguments x, y into their father and mother, respectively. For example, a person named *Elizabeth* is the mother of *Charles*, who is the father of *William*.

We would like to establish whether the formula $f(x_1) = x_2 \wedge m(x_2) = x_3$ is true under interpretation $\mathcal{M}$ with respect to an object assignment. It depends on how variables x_1, x_2, x_3 are instantiated by people.

If the object assignment σ' assigns all variables to the same element *William*, this formula is false since William is not the mother of the father of himself.

However, if $\sigma(x_1) = William^{\mathcal{M}}$, $\sigma(x_2) = Charles^{\mathcal{M}}$, $\sigma(x_3) = Elizabeth^{\mathcal{M}}$, and all other variables are assigned distinct people, then $\mathcal{M}$ interpretation of LHS under σ is $\big(f(x_1)\big)^{\mathcal{M}} = f^{\mathcal{M}}(x_1)[\sigma] = f^{\mathcal{M}}(William^{\mathcal{M}})[\sigma] = Charles^{\mathcal{M}}[\sigma] = x_2[\sigma]$. Also, $(\ m(x_2)\ )^{\mathcal{M}} = m^{\mathcal{M}}(x_2)[\sigma] = m^{\mathcal{M}}(Charles^{\mathcal{M}})[\sigma] = Elizabeth^{\mathcal{M}}[\sigma] = x_3[\sigma]$. Therefore, with this 2nd object assignment, the given formula is true since σ maps x_3 to the person named Elizabeth.

Basic Semantic Definition: Motivation

Our plan is the following. We are going to define for an arbitrary wff F in a language L whether it holds under an L-interpretation $\mathcal{M}$ when σ assigns elements from the universe D of $\mathcal{M}$ to all variables. Notation: $\mathcal{M} \models F[\sigma]$.

Subsequently, similar to propositional logic, we are interested in solving the logical consequence problem for first order logic (FOL), i.e., whether a *QueryFOL* formula is entailed from the given knowledge base *kbFOL*, or in symbols, $kbFOL \models^{?} QueryFOL$. The following definition parallels similar definition in propositional logic.

Definition. We say that a FOL formula F is a *logical consequence* of a set of formulas KB, in symbols, $KB \models F$, iff for all interpretations $\mathcal{M}$ and *all object assignments* σ, if $\mathcal{M}$ is a model for KB, $\mathcal{M} \models KB$, then $\mathcal{M}$ satisfies F: $\mathcal{M} \models F$

Similar to propositional logic, this logical consequence problem is equivalent to deciding whether the conjunction of $(\bigwedge KB) \wedge \{\neg F\}$ is unsatisfiable. We are going to discuss later how to use an extended tableau algorithm that deals with quantifiers to solve (sometimes) the unsatisfiability problem.

With this foresight in mind, we proceed to define precisely when $\mathcal{M}$ is a *model* for F under object assignment σ. *First*, we define recursively over structure of a term how terms are interpreted. *Second*, we define recursively over structure of a formula F, when $\mathcal{M} \models F[\sigma]$.

(BSD) Basic Semantic Definition: (1) for terms

Since terms are strings built out of sub-terms by applying a function symbol to sub-terms, the meaning of terms is defined recursively similar to how term formation trees are defined. Since terms may contain variables, they are interpreted with respect to (w.r.t.) an object assignment σ.

Definition

Let $\mathcal{M}$ be an L-structure, and let σ be an object assignment for $\mathcal{M}$. Each term t is assigned an element $t^{\mathcal{M}}[\sigma]$ in D with respect to σ as follows.

- If term t is a constant C, then $C^{\mathcal{M}}$ is the interpretation of this constant under $\mathcal{M}$.
- For each variable x, if term t is a variable x, then $x^{\mathcal{M}}[\sigma]$ is an element mapped by object assignment $\sigma(x)$.
- For any n-place function symbol f in L, whose arguments are terms $\langle t_1, \ldots, t_n\rangle$, interpretation of the term $f(t_1, \ldots, t_n)$ under σ, or in symbols $(f(t_1, \ldots, t_n))^{\mathcal{M}}[\sigma]$, is an element mapped by the function interpreting f applied to interpretation of arguments w.r.t. σ, or in symbols, $f^{\mathcal{M}}(t_1^{\mathcal{M}}[\sigma], \ldots, t_n^{\mathcal{M}}[\sigma])$.

If t is a closed term, i.e., it contains no variables, then $t^{\mathcal{M}}[\sigma]$ is independent of σ and so we can use a shorter notation $t^{\mathcal{M}}$.

(BSD) Basic Semantic Definition: (2) for formulas

Recall we associate with σ a family of object assignments $\sigma(x \mapsto d)$: for each $d \in D$ we consider $\sigma(x \mapsto d)$ that maps x into d, but otherwise it is like σ. For any string in L that is a wff F, we say that $\mathcal{M}$ satisfies F under σ, or in symbols $\mathcal{M} \models F[\sigma]$, in the following cases.

(a). $\mathcal{M} \models P(t_1, \ldots, t_n)[\sigma]$ iff $\langle t_1^{\mathcal{M}}[\sigma], \ldots, t_n^{\mathcal{M}}[\sigma]\rangle \in P^{\mathcal{M}}$, i.e., an atomic formula with a predicate symbol P is satisifed by $\mathcal{M}$ under σ iff the tuple of its arguments belongs to relation $P^{\mathcal{M}}$ interpreting P.

(b). $\mathcal{M} \models (s = t)[\sigma]$ iff $s^{\mathcal{M}}[\sigma] = t^{\mathcal{M}}[\sigma]$. Note this follows from the previous item and the fact that $=^{\mathcal{M}}$ is always the equality relation.

(c). $\mathcal{M} \models \neg F[\sigma]$ iff not $\mathcal{M} \models F[\sigma]$, i.e., an interpretation $\mathcal{M}$ satisfies $\neg F$ under σ iff $\mathcal{M}$ does **not** satisfy F under σ. Note that either $\mathcal{M} \models F[\sigma]$ or $\mathcal{M} \models \neg F[\sigma]$.

(d). $\mathcal{M} \models (F_1 \vee F_2)[\sigma]$ iff $\mathcal{M} \models F_1[\sigma]$ or $\mathcal{M} \models F_2[\sigma]$.

(e). $\mathcal{M} \models (F_1 \wedge F_2)[\sigma]$ iff $\mathcal{M} \models F_1[\sigma]$ and $\mathcal{M} \models F_2[\sigma]$.

(f). $\mathcal{M} \models (\forall x F)\,[\sigma]$ iff $\mathcal{M} \models F\,[\sigma(x \mapsto d)]$ for <u>all</u> elements $d \in D$.

(g). $\mathcal{M} \models (\exists x F)\,[\sigma]$ iff $\mathcal{M} \models F\,[\sigma(x \mapsto d)]$ for <u>some</u> element $d \in D$.

(h). For $(F_1 \rightarrow F_2), (F_1 \leftrightarrow F_2)$ use their definitions.

Basic Semantic Definition: Examples

Notice that if a wff F is a sentence, i.e., it has no free variables, then sometimes we write $\mathcal{M} \models F$ instead of $\mathcal{M} \models F[\sigma]$, since the object assignment σ does not matter in this case.

Example. Consider language L that includes only two binary predicate symbols $\{\preceq, =\}$, but has no function symbols. Interpret L-formulas in L-structure $\mathcal{M}$ whose universe D is the set of positive integer numbers and such that $\preceq^{\mathcal{M}}(m, n)$ iff $m \leq n$. Consider the following formulas, and explain whether $\mathcal{M}$ satisfies them or not, and why.

- $\mathcal{M} \models^? \exists x \forall y (x \preceq y)$
- $\mathcal{M} \models^? \exists y \forall x (x \preceq y)$
- $\mathcal{M} \models^? \forall x \exists y \exists z(\,(x \preceq y) \wedge (y \preceq z)\,)$
- $\mathcal{M} \models^? \forall x \forall y \forall z((x \preceq y) \wedge (y \preceq z)) \rightarrow (x \preceq z)$

The standard model $\mathcal{N}$ for arithmetics has non-negative integers as universe, $s^{\mathcal{N}}(n)$ is $n+1$, function symbols $0, \cdot, +$ get their usual meanings. Then, $\mathcal{N} \models \forall x \forall y \exists z (x{+}z{=}y \vee y{+}z{=}x)$ and $\mathcal{N} \models \forall x \forall y \forall z(\,(x+y)\cdot z = (x \cdot z + y \cdot z)\,)$.

But $\mathcal{N} \not\models \forall x \exists y (y + y = x)$ and $\mathcal{N} \not\models \forall x \exists y \exists z (y \cdot z = x \rightarrow \neg(y = s(0)) \wedge \neg(y = x)\,)$.

FOL: satisfiability, validity, equivalences

We use KB to denote a set of formulas in a knowledge base, F_1, F_2, G to denote arbitrary wff in FOL. $\mathcal{M}$ denotes a structure, σ is object assignment.

Definition F is *satisfiable* iff $\mathcal{M} \models F[\sigma]$ for some $\mathcal{M}$ and σ.

Definition $\mathcal{M} \models KB[\sigma]$ iff $\mathcal{M} \models F[\sigma]$ for all $F \in KB$. (We omit σ if KB is a set of sentences.) Say KB is *satisfiable* if $\mathcal{M} \models KB[\sigma]$ for some $\mathcal{M}$ and σ.

Definition F is *valid*, in symbols, $\models F$, iff $\mathcal{M} \models F[\sigma]$ for all $\mathcal{M}$ and σ.

Definition F_1 and F_2 are *logically equivalent*, in symbols, $F_1 \equiv F_2$, iff for all $\mathcal{M}$ and σ both F_1 and F_2 are satisfiable by $\mathcal{M}$ under σ at the same time.

Definition (logical consequence) $KB \models F$ iff for all $\mathcal{M}$ and σ, if $\mathcal{M} \models KB[\sigma]$, then $\mathcal{M} \models F[\sigma]$. (Usually, both KB and F are sentences, so we can omit σ.)

Caution: the symbols "$\models$" and "$\equiv$" are symbols of the meta-langauge, as opposed to symbols $\neg, \wedge \vee, \forall, \exists$ which are symbols that can occur in formulas. Write "$\models$" and "$\equiv$" only between formulas, never inside formulas.

Notation: if KB is a finite set of formulas $\{F_1, F_2, \ldots, F_n\}$, then we sometimes write $F_1, F_2, \ldots, F_n \models G$ instead of $\{F_1, F_2, \ldots, F_n\} \models G$.

Establish logical consequence using BSD

Example
Prove $(\forall x F_1 \vee \forall x F_2) \models \forall x (F_1 \vee F_2)$ for any wffs F_1, F_2 from the definition of logical consequence: $KB \models F$ iff for all $\mathcal{M}, \sigma$, if $\mathcal{M} \models KB[\sigma]$, then $\mathcal{M} \models F[\sigma]$.

Let $\mathcal{M}$ be any interpretation, and let σ be any object assignment. Assume L.H.S. is true, i.e. $\mathcal{M} \models (\forall x F_1 \vee \forall x F_2)[\sigma]$.

Then, from BSD(d), either $\mathcal{M} \models (\forall x F_1)\,[\sigma]$ or $\mathcal{M} \models (\forall x F_2)\,[\sigma]$.

Case 1: $\mathcal{M} \models (\forall x F_1)\,[\sigma]$. Then, according to BSD(f), for all $d \in D$, we have that $\mathcal{M} \models F_1[\sigma(x \mapsto d)]$. Since $\mathcal{M}$ satisfies F_1, it satisifies the disjunction.

Therefore, for all $d \in D$ we have $\mathcal{M} \models (F_1 \vee F_2)[\sigma(x \mapsto d)]$.

Consequently, according to BSD(f), $\mathcal{M} \models \forall x (F_1 \vee F_2)[\sigma]$, i.e., R.H.S. holds.

Case 2: $\mathcal{M} \models (\forall x F_2)\,[\sigma]$ is very similar. It also leads to R.H.S. is true.

Counterexample to logical consequence

Example
What can we say about logical consequence in the opposite direction ?

$$\forall x (F_1 \vee F_2) \models^? (\forall x F_1 \vee \forall x F_2), \text{ for any wffs } F_1, F_2.$$

To establish logical consequence we have to use BSD. But, to prove lack of logical consequence, it is sufficient to show a counterexample. This is because $(LHS) \not\models (RHS)$, if there is a model for LHS that does not satisfy RHS. So, we invent an interpretation satisfying LHS but falsifying RHS.

Consider the domain of all people. Let both F_1, F_2 be atomic formulas constructed from 1-place predicates. Interpret $F_1(x)$ as saying "x is male", and interpret $F_2(x)$ as saying "x is female".

Then, obviously, L.H.S. is true, since under this interpretation, every person is either male or female.

However, R.H.S. is false since it is false that either all people are male, or all people are female.

A different counterexample can be constrcuted if we consider positive integer numbers as universe, and interpret $F_1(x)$ as "x is odd", and $F_2(x)$ as "x is even". Every integer is even or odd, but it is not the case that all integers are even, or all integers are odd.

Establish logical consequence using BSD

Example
Using BSD and definition of logical consequence, prove that for any wff F

$$\neg(\exists x F(x)) \models \forall x(\neg F(x)).$$

Let $\mathcal{M}$ be any structure and let σ be any object assignment. Suppose L.H.S is true, i.e., $\mathcal{M} \models \neg(\exists x F(x))\,[\sigma]$.

By BSD(c), it is **not** the case that $\mathcal{M} \models (\exists x F(x))\,[\sigma]$ since if negation of a formula is satisifed, then formula itself is not.

By BSD(g), it is not the case that for **some** element d of the domain the formula $F(x)$ is satisfied under object assignment $[\sigma(x \mapsto d)]$, or in symbols, $\mathcal{M} \models F(x)\,[\sigma(x \mapsto d)]$.

Therefore, for **all** elements d it is not the case that $\mathcal{M} \models F(x)\,[\sigma(x \mapsto d)]$.

Then, by BSD(c), for **all** elements d in the domain we have that $\mathcal{M} \models \neg F(x)\,[\sigma(x \mapsto d)]$.

From BSD(f) follows that $\mathcal{M} \models \forall x(\neg F(x))\,[\sigma]$. So, we have proved R.H.S.

The opposite direction $\forall x(\neg F(x)) \models \neg(\exists x F(x))$ also holds and can be similarly proved. Therefore, $\forall x(\neg F(x)) \equiv \neg(\exists x F(x))$.

Logically equivalent formulas: Part 1

Let $F(x), G(x)$ be FOL well-formed formulas with a free variable x. Let B be a FOL well-formed formula without free occurrences of x. Let $A(x, y)$ be a FOL wff with free variables x and y.
The following equivalences can be proved from BSD. The formulas with implication are valid. Note that if $\models (F \to G)$, then $F \models G$.

$\forall x F(x) \equiv \neg\exists x \neg F(x)$	$\exists x F(x) \equiv \neg\forall x \neg F(x)$
$\forall x \forall y\, A(x,y) \equiv \forall y \forall x\, A(x,y)$	$\exists x \exists y\, A(x,y) \equiv \exists y \exists x\, A(x,y)$
$\exists x \forall y\, A(x,y) \to \forall y \exists x\, A(x,y)$	/* one direction; to be discussed soon */
$(\exists x F(x) \lor B) \equiv \exists x(F(x) \lor B)$	$(\forall x F(x) \lor B) \equiv \forall x(F(x) \lor B)$
$(B \lor \exists x F(x)) \equiv \exists x(B \lor F(x))$	$(B \lor \forall x F(x)) \equiv \forall x(B \lor F(x))$
$(\exists x F(x) \land B) \equiv \exists x(F(x) \land B)$	$(\forall x F(x) \land B) \equiv \forall x(F(x) \land B)$
$(B \land \exists x F(x)) \equiv \exists x(B \land F(x))$	$(B \land \forall x F(x)) \equiv \forall x(B \land F(x))$
If $KB \models B$ then $\models (KB \to B)$	/* similar to propositional logic */

Logically equivalent formulas: Part 2

Let $F(x), G(x)$ be FOL well-formed formulas with a free variable x. Let A, B be FOL well-formed formulas without free occurrences of x. The following equivalences can be proved directly from BSD.

$\forall x(A \to G(x)) \equiv (A \to \forall x G(x))$
$\forall x(F(x) \to B) \equiv (\exists x F(x) \to B)$

$(\exists x(F(x) \lor G(x)) \equiv (\exists x F(x) \lor \exists x G(x))$
$\forall x(F(x) \land G(x)) \equiv (\forall x F(x) \land \forall x G(x))$

$\forall x F(x) \lor \forall x G(x)) \to \forall x(F(x) \lor G(x))$ /* one direction only, why? */
$\exists x(F(x) \land G(x)) \to (\exists x F(x) \land \exists x G(x))$ /* one direction only, why? */

$\forall x(F(x) \leftrightarrow G(x)) \to (\forall x F(x) \leftrightarrow \forall x G(x))$ /* one direction only */
$\forall x(F(x) \leftrightarrow G(x)) \to (\exists x F(x) \leftrightarrow \exists x G(x))$ /* one direction only */

$\exists x(F(x) \to G(x)) \equiv (\forall x F(x) \to \exists x G(x))$
$(\exists x F(x) \to \forall x G(x)) \to \forall x(F(x) \to G(x))$ /* one direction only */

Formulas with Implication in One Direction only

We can easily prove $\exists x(F(x) \to G(x)) \equiv (\forall x F(x) \to \exists x G(x))$ by doing logically equivalent transformations., e.g., from LHS to RHS.
$\exists x(F(x) \to G(x)) \equiv \exists x(\neg F(x) \lor G(x)) \equiv (\exists x \neg F(x) \lor \exists x G(x)) \equiv \neg\forall x\, F(x) \lor \exists x\, G(x)) \equiv (\forall x\, F(x)) \to (\exists x\, G(x))$.

Exercises. Explain briefly why implication would not work in the opposite direction: provide a counter-example.

$\forall x(F(x) \lor G(x)) \to (\forall x F(x) \lor \exists x G(x))$
$\forall x(F(x) \to G(x)) \to (\forall x F(x) \to \forall x G(x))$
$\forall x(F(x) \to G(x)) \to (\exists x F(x) \to \exists x G(x))$
$\forall x(F(x) \to G(x)) \to (\forall x F(x) \to \exists x G(x))$

Substitution in terms and in formulas

Definition Let u, t be terms. Notation $t(u/x)$ is the result of replacing all occurrences of x in t by u. Notation $F(u/x)$ is the result of replacing all *free* occurrences of x in F by u.

Theorem For each interpretation $\mathcal{M}$ and each object assignment σ,

$$(t(u/x))^{\mathcal{M}}[\sigma] = t^{\mathcal{M}}[\sigma(x \mapsto m)],$$

where m is a domain element representing $u^{\mathcal{M}}[\sigma]$.

Warning: substitution in formulas can go wrong. Let F be $\forall y \neg(x = y + y)$. This says "$x$ is odd". But substitution $F(x + y/x)$ is $\forall y \neg(x + y = y + y)$. This does not say "$x + y$ is odd" as desired, but instead for any instantiation of the variable x, it is false. To see why, consider its equivalent rewriting $\neg\exists y(x + y = y + y)$ and notice if $x = y$, then $(x + y = y + y)$.
The problem is that y in the term $x + y$ got caught by the quantifier $\forall y$. To avoid this rename y to a fresh variable z. This motivates an extra condition.

Definition A term t is *freely substitutable* for x in F iff for every variable y that occurs in t no free occurrence of x in F is in a subformula of F of the form $\forall yG$ or $\exists yG$. (Note to avoid collisions all bound variables can be renamed.)

Theorem If t is freely substitutable for x in wff F, then for all interpretations $\mathcal{M}$ and all object assignments σ, $\mathcal{M} \models F(t/x)[\sigma]$ iff $\mathcal{M} \models F[\sigma(x \mapsto m)]$, where m is a domain element representing $t^{\mathcal{M}}[\sigma]$.

References

Mordechai Ben-Ari "Mathematical Logic For Computer Science", the 3rd edition (2012), Springer. (Chapter 2 Propositional Logic: Sections 2.1–2.5 and Chapter 7 First-Order Logic: Sections 7.1–7.4.) `https://link.springer.com/book/10.1007/978-1-4471-4129-7`

Stephen Cook CSC 438F/2404F: Computability and Logic (University of Toronto), `http://www.cs.toronto.edu/~sacook/csc438h/`

Herbert Enderton: A Mathematical Introduction to Logic. Academic Press, 1972. It is recommended to read Chapter 2: Sections 2.1–2.6 `https://archive.org/download/MathematicalIntroductionToLogicEnderton/MathematicalIntroductionToLogic-Enderton.pdf`

Elliot Mendelson "Introduction to Mathematical Logic", 4th edition, published by Chapman Hall, 1997. Chapter 2: Sections 2.1 – 2.3 are recommended.

Uwe Schoning "Logic for Computer Scientists", published by Birkhauser, Boston, 2008, ISBN 978-0-8176-4763-6. It is recommended to read Sections 1.1 and 1.2 from Chapter 1 "Propositional Logic" and Section 2.1 and 2.2 from Chapter 2.