First-Order Predicate Logic

The classical motivating example:

All humans are mortal.
Socrates is human.
Therefore:
Socrates is mortal.

- This cannot be expressed in propositional logic.
- Propositions with variables are needed:

For all objects h:
$\text{Human}(h) \rightarrow \text{Mortal}(h)$.

Thus, substituting Socrates for h:
$\text{Human}(\text{Socrates}) \rightarrow \text{Mortal}(\text{Socrates})$.

Since Socrates is human, the following may be asserted:
$\text{Human}(\text{Socrates})$.

Modus ponens may now be applied to deduce

$\text{Mortal}(\text{Socrates})$.

We write “for all” as $\forall$, so

$$((\forall h)(\text{Human}(h) \rightarrow \text{Mortal}(h)) \wedge \text{Human}(\text{Socrates})) \rightarrow \text{Mortal}(\text{Socrates})$$

Further motivation: Some examples from the blocks world:

No object can rest atop another object and lie on the table at the same time.:

$$(\forall x)(\forall y)\ (\neg(\text{On\_table}(x) \wedge \text{On}(x,y)))$$

If the number of objects is finite (as it is in our blocks world), then this may be expressed using propositional logic. However, with four objects, there are sixteen possibilities, which must be conjoined to realize an equivalent statement.

$$\begin{array}{l}
(\neg(\text{On\_table}[B1] \wedge \text{On}[B1,B1])) \wedge \\
(\neg(\text{On\_table}[B1] \wedge \text{On}[B1,B2])) \wedge \\
(\neg(\text{On\_table}[B1] \wedge \text{On}[B1,P1])) \wedge \\
(\neg(\text{On\_table}[B1] \wedge \text{On}[B1,P2])) \wedge \\
(\neg(\text{On\_table}[B2] \wedge \text{On}[B2,B1])) \wedge \\
(\neg(\text{On\_table}[B2] \wedge \text{On}[B2,B2])) \wedge \\
(\neg(\text{On\_table}[B2] \wedge \text{On}[B2,P1])) \wedge \\
(\neg(\text{On\_table}[B2] \wedge \text{On}[B2,P2])) \wedge \\
(\neg(\text{On\_table}[P1] \wedge \text{On}[P1,B1])) \wedge \\
(\neg(\text{On\_table}[P1] \wedge \text{On}[P1,B2])) \wedge \\
(\neg(\text{On\_table}[P1] \wedge \text{On}[P1,P1])) \wedge \\
(\neg(\text{On\_table}[P1] \wedge \text{On}[P1,P2])) \wedge \\
(\neg(\text{On\_table}[P2] \wedge \text{On}[P2,B1])) \wedge \\
(\neg(\text{On\_table}[P2] \wedge \text{On}[P2,B2])) \wedge \\
(\neg(\text{On\_table}[P2] \wedge \text{On}[P2,P1])) \wedge \\
(\neg(\text{On\_table}[P2] \wedge \text{On}[P2,P2]))
\end{array}$$

Clearly, this form of propositional representation can get out of hand in a hurry.

Every object is either on the table or else atop another object.

Here we must use the “there exists” connective, which is abbreviated $\exists$:

$$(\forall x)(\exists y)(\text{On\_table}(x) \vee \text{On}(x,y))$$
or
$$(\forall x)(\text{On\_table}(x) \vee (\exists y)\text{On}(x,y))$$

To expand this to propositional logic, for each x, the alternatives for y must be disjoined.

$$\begin{array}{l}
(\ \text{On\_table}[B1] \vee \\
\quad \text{On}[B1,B1] \vee \text{On}[B1,B2] \vee \\
\quad \text{On}[B1,P1] \vee \text{On}[B1,P2]) \wedge \\
(\ \text{On\_table}[B2] \vee \\
\quad \text{On}[B2,B1] \vee \text{On}[B2,B2] \vee \\
\quad \text{On}[B2,P1] \vee \text{On}[B2,P2]) \wedge \\
(\ \text{On\_table}[P1] \vee \\
\quad \text{On}[P1,B1] \vee \text{On}[P1,B2] \vee \\
\quad \text{On}[P1,P1] \vee \text{On}[P1,P2]) \wedge \\
(\ \text{On\_table}[P2] \vee \\
\quad \text{On}[P2,B1] \vee \text{On}[P2,B2] \vee \\
\quad \text{On}[P2,P1] \vee \text{On}[P2,P2])
\end{array}$$

Again, this propositional representation leads to a combinatorial explosion of notation.

General notions of first-order predicate logics:

In comparison to a propositional logic, a first-order predicate logic has the following features:

- The proposition names can take arguments. Thus, instead of a simple proposition name A, we will work with statements which look like

  $$A(t_1, t_2, .., t_n)$$

  in which the t_i’s are arguments to A. In this context, A is called a *relation symbol*, and the number of arguments which it takes (n in this example) is called its *arity.* It is customary to use upper-case letters such as P, Q, R, S, and T, possibly with subscripts, as relation symbols, although letters such as A, B, and C may be used as well, as may more descriptive names, such as those used in the blocks-world example:

Relation Symbol	Arity
On	2
On_table	1
Is_cube	1
Is_pyramid	1

- Each relation symbol has exactly one arity; there is no polymorphism in the first-order logics discussed in these notes.

- The arguments of the relation symbols are called *terms*. The set of terms includes at least the following:

  - A countably infinite set V of variables. Usually, lower-case letters from the end of the alphabet, possibly with subscripts, will be used to denote variables; *e.g.,* x, y, x_1, z_{12}. In the blocks world example, x, y, and z were used a variables.

  - A set K of constant symbols. This set may be empty, but in practical applications in computer science it almost never is.

  In the blocks world example, the set of constant symbols consists of {B1, B2, P1, P2}.

- A set of *function symbols*, which may be applied recursively.

The blocks-world example did not include any function symbols. However, we could augment the existing description with some simple functions.

Consider the functions *other* and *opposite*, defined informally as follows:

x	other(x)	opposite(x)
B1	B2	P1
B2	B1	P2
P1	P2	B1
P2	P1	B2

In words, Other identifies the other block of the same type, and Opposite identifies the block of the opposite type, but with the same index.

We may thus write formulas such as:

(∃x)On(x, other(x))
(∃x)On(other(x),opposite(x))
(∀x)(∀y)((¬On(x,other(opposite(y)))

These functions are always *total;* they may not be undefined for any arguments.

These functions may be composed indefinitely:

On(x,other(other(other(other(opposite(x)))))))

In addition to the connectives of propositional logic, there are two new *quantifiers:*

- The *universal quantifier* ∀. ("for all")

- The *existential quantifier* ∃. ("there exists")

Use of these has already been illustrated in the blocks world.

(∀x)(∀y) (¬(On_table(x) ∧ On(x,y)))

(∀x)(∃y)(On_table(x) ∨ On(x,y))

We will see that either one may be defined in terms of the other.

- An extremely important relation within first-order predicate logic is equality. It occurred in several of the sentences in the blocks-world example.

  An object can rest atop at most one object.
  (∀x)(∀y)(∀z) ((On(x,y) ∧ On(x,z)) → y=z))

  At most one object can rest atop another object.
  (∀x)(∀y)(∀z) ((On(y,x) ∧ On(z,x)) → y=z))

  Equality is an extremely important operation. However, it also complicates the presentation of a logic substantially, and so it will be omitted from the initial presentation. It will be introduced in due course.

  Unfortunately, the textbook does not develop the notion of equality within first-order predicate logic at all.

There are many flavors of first-order predicate logic.

We will look at several of the most common.

- Function-free first-order logic.
  This is the simplest form. It will be studied first, in order to gain familiarity with the basic notions while incurring the least amount of overhead from new ideas. Despite its simplicity, it has substantial application in computer science, for example, in the formulation of knowledge in deductive databases.

- First-order logic with functions.
  This is the most common, general form. It is the one which the textbook uses throughout.

- First-order logic with equality.
  This is an extremely important class of first-order logics, which allows us to write sentences using the equality predicate "=", such as
  An object can rest atop at most one object:
  $(\forall x)(\forall y)(\forall z)((On(x,y) \wedge On(x,z)) \rightarrow y=z)$

  Unfortunately, this logic is not developed in the textbook at all, although it is used in some examples.

Term Algebras:

Definition: A *function-free term algebra* $T = (V, K)$ consists of the following:

- A countable set V of *variables.*
- A (possibly empty) set K of *constant symbols* or *constant names.*

It is always assumed that $V \cap K = \emptyset$.

T is called *finite* if K is a finite set.

Example: Let K_{BW} = {B1, B2, P1, P2}, and let $V = \{x_0, x_1, x_2, \ldots\}$. Then $T_{BW} = (V, K_{BW})$ is a function-free term algebra for the blocks world. Note that it is finite.

Remark: Except in the instance of some very formal definitions and proofs, it is customary to assume that V consists of lower case letters from the end of the alphabet, as well as such letter subscripted: x, y, z, x_1, y_{10}, z_a.

A *function-free term* over $T = (V, K)$ is just an element of $V \cup K$. The set of all such terms is denoted Terms(T).

Definition: A full term algebra (or just term algebra) is a triple $T = (V, K, F)$ subject to the following conditions.

- (V, K) is a function-free term algebra.

- F is a set of *non-nullary function symbols* (or just *function symbols*).

- With each $f \in F$ is a associated a positive integer Arity(f), called the *arity* of f. Informally, Arity(f) identifies the number of arguments which f takes.

The *terms* over T, denoted Terms(T), are defined inductively as follows.

(a) Each $v \in V$ is in Terms(T).
(b) Each $k \in K$ is in Terms(T).
(c) If $t_1, t_2, .., t_n \in$ Terms(T), and if $f \in F$ with Arity(f) = n, then $f(t_1,t_2,..,t_n) \in$ Terms(T).

Definition: A *first-order logic* is a quadruple $L = (R, C, A, T)$ in which:

- R is a set of *relation names*. With each such name $R \in R$ is associated a natural number Arity(R), called the *arity* of R.
- $C = \{\neg, \rightarrow, \bot, \forall\}$ is the set of *logical connectives.*
- A = { (,) } is the set of auxiliary symbols.
- $T = (V, K, F)$ is a term algebra.

L is termed *function free* if T is function free.

It is assumed that the names in these sets are such that there are no collisions. For example, $\forall$ cannot be a constant symbol.

Example: Let R be as defined in the following table.

Relation Symbol	Arity
On	2
On_table	1
Is_cube	1
Is_pyramid	1

And let $T_{BW} = (V, K_{BW})$ be the term algebra for the blocks world defined previously. Then $L_{BW} = (R_{BW}, C, A, T_{BW})$ is a function-free first-order logic for this world.

Semantics:

In propositional logic:

- an interpretation provides a truth value for each proposition.

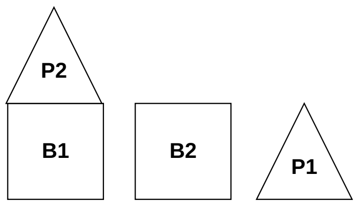
In first-order logic:

- The “propositions” are relation symbols which take arguments.
- Thus, there is not a single truth value.
- An interpretation identifies those arguments (tuples) for which the relation is true, and those for which it is false.

Definition: Let L = (R, C, A, T) be a first-order logic, with T = (V, K, F). An interpretation J for L consists of the following:

- A set Dom(J), called the *domain* of the interpretation. This set need not be finite.
- For each relation symbol R, a relation R^J of Arity(R) columns over Dom(J).
- For each constant symbol $K \in K$, an associated element K^J of Dom(J).
- For each function symbol $f \in F$ of arity n, an n-ary function f^J: $Dom(J)^n \rightarrow Dom(J)$.

Example: Here is an interpretation J describing the state of the blocks world which is shown below.



D = {d_1, d_2, d_3, d_4}

Item	**Interpretation in J**
$B1^J$	d_1
$B2^J$	d_2
$P1^J$	d_3
$P2^J$	d_4
On_table^J	{ (d_1), (d_2), (d_3) }
On^J	{ (d_4, d_1) }
Is_cube^J	{ (d_1), (d_2) }
$Is\_pyramid^J$	{ (d_3), (d_4) }

Note:

- Constant symbols are not in the interpretation. It is not correct to write On^J = { (P2,B1) }.
- The names of the elements in D is not important.
- Any set with four or more elements could serve as the domain. Not all domain elements need be “used.”

Suppose that we wish to extend this interpretation to include the functions *other* and *opposite* defined earlier.

The chart presented earlier

x	other(x)	opposite(x)
B1	B2	P1
B2	B1	P2
P1	P2	B1
P2	P1	B2

is a bit deceptive. The interpretation acts on domain elements, not constant symbols. For the interpretation on the previous slide, this chart should read:

X	other(x)	opposite(x)
d_1	d_2	d_3
d_2	d_1	d_4
d_3	d_4	d_1
d_4	d_3	d_2

The assignment of domain values to constant symbols effectively defines the first chart above, however.

Well-Formed Formulas:

In first-order logic, the fundamental object, analogous to a proposition in propositional logic, is called an atom. It is defined as follows.

Definition: Let L = (R, C, A, T) be a first-order logic. Let $t_1, t_2, .., t_n \in$ Terms(T), and let $R \in R$ with Arity(R) = n.
Then $R(t_1,t_2,..,t_n)$ is termed an *atomic formula,* or *atom*, of L. The set of all such atoms is denoted Atoms(L).

Note: An atom is just an uninterpreted string of symbols, and is not to be viewed as a function applied to arguments.

Examples: In the logic L_{BW}, On(x_1,x_2), On(P1,x_1), and On_table(P1) are all examples of atoms.

Definition: A *ground atom* is one in which no term involves a variable.

Examples: Of the above three, only On_table(P1) is a ground atom. On(P1,B1) is another example of a ground atom.

Important: It must be emphasized that terms are not wff’s. Terms are arguments to relations, and do not themselves have truth values.

Definition: Let $L = (R, C, A, T)$ be a first-order logic. The class of *(strict) well-formed formulas* over L (denoted SWF(L)), is the smallest set of strings which is closed under the following rules:

(a) $\bot \in$ SWF(L).

(b) If $t_1, t_2, .., t_n \in$ Terms(T), and $R \in R$ with Arity(R) = n, then $R(t_1,t_2,..,t_n) \in$ SWF(L).

(c) $\varphi \in$ SWF(L) implies that $(\neg\varphi) \in$ SWF(L).

(d) $\varphi_1, \varphi_2 \in$ SWF(L) implies that
$$(\varphi_1 \to \varphi_2) \in \text{SWF}(L).$$

(e) $\varphi \in$ SWF(L) and $v \in V$ implies that
$$(\forall v)(\varphi) \in \text{SWF}(L).$$

Notes:

- A formula of the form $R(t_1,t_2,..,t_n)$ is called an *atom.*

Example: In the logic L_{BW}:

- On(x_1,x_2), On(P1,x_1), and On_table(P1) are all examples of atoms.

$$(\forall x_1)(\forall x_2)(\forall x_3)\ (On(x_1,x_3) \to (\neg On\_table(x_2)))$$

is an example of a strict well-formed formula.

All of the following abbreviations, except the last, is familiar from propositional logic.

Definitions:
$(\varphi_1 \vee \varphi_2)$ is an abbreviation for $((\neg\varphi_1) \to \varphi_2)$.
$(\varphi_1 \wedge \varphi_2)$ is an abbreviation for $(\neg(\varphi_1 \to (\neg\varphi_2)))$.
$(\varphi_1 \equiv \varphi_2)$ is an abbreviation for
$$((\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)).$$
$\top$ is an abbreviation for $(\neg\bot)$.
$(\exists x)(\varphi)$ is an abbreviation for $(\neg(\forall x)(\neg \varphi)))$

The last abbreviation is easily understood by considering a simple example. Compare the following pairs of statements, and convince yourself that they mean the same thing.

$$(\exists x)On(P1,x)$$
and
$$(\neg(\forall x)(\neg On(P1,x)))$$

$$(\forall x)On\_table(x)$$
and
$$(\neg(\exists x)(\neg On\_table(x)))$$

As with propositional logic, we let WF(L) denote the class of all well-formed formulas over L. This class includes the elements of SWF(L), as well as formulas involving the abbreviations identified above. From now on, we will use wff or wf as an abbreviation for well-formed formula.

The scope of quantifiers:

The truth value of formulas such as

$$(\exists x)On(P1,x)$$

$$(\forall x)(On\_table(x) \vee (\exists y)On(x,y))$$

Is understandable in fairly intuitive terms. However, others may not be so simple. Consider, for example:

$$(\forall x)((Is\_cube(x) \wedge On\_table(x))) \to (\exists y)On(y,x)$$

$$(\forall x)(\exists y)((Is\_cube(x) \wedge On\_table(x))) \to On(y,x)$$

$$(\forall x)((Is\_cube(x) \wedge On\_table(x)) \to (\exists x)On(x,x))$$

$$(\forall x)((Is\_cube(x) \wedge On\_table(x)) \to On(y,x))$$

$$On(P1,x)$$

Such formulas are legal under the definitions provided, and we must establish their meanings.

The *scope* of a quantifier is the part of the formula immediately to its right, excepting that part which is scoped by a quantifier of the same name. This is most easily illustrated by example.

$$(\forall x)\overbrace{((Is\_cube(x) \wedge On\_table(x)) \to (\exists y)\underbrace{On(y,x)}_{\text{Scope of y}})}^{\text{Scope of x}}$$

$$(\forall x)(\exists y)\overbrace{\underbrace{((Is\_cube(x) \wedge On\_table(x)) \to On(y,x))}_{\text{Scope of y}}}^{\text{Scope of x}}$$

$$(\forall x)\overbrace{((Is\_cube(x) \wedge On\_table(x)) \to (\exists x)}^{\text{Scope of outer x}}\underbrace{On(x,x))}_{\text{Scope of inner x}}$$

With regards to using the same variable more than once, the situation is identical to that of lexically scoped programming languages. For example, in a the following program

```
        procedure outer
           var x;
                 …
           procedure inner
              var x:
              begin
                 x := x+1
              end; {inner}
                 …
        end: {outer}
```

the variable x declared in procedure `outer` is not visible in procedure `inner.` It is said to be *shadowed* by the inner variable.

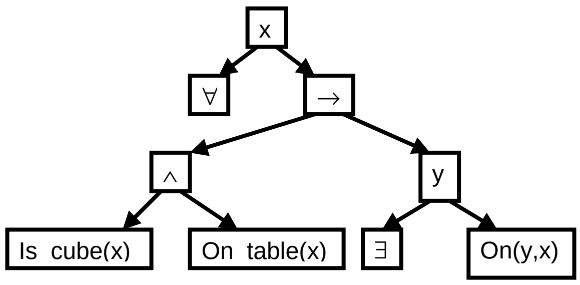
Similarly, in the following formula, the variable x which is universally quantified is shadowed by the one which is existentially quantified.

$$(\forall x)((\text{Is\_cube}(x) \wedge \text{On\_table}(x)) \rightarrow (\exists x)\text{On}(x,x))$$

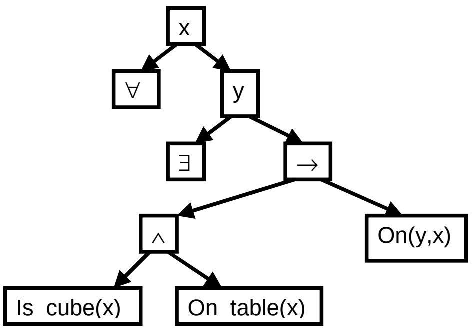
In programming languages, the advisability of shadowing is debatable. In logical formulas, it is always to be avoided.

Scoping may also be represented using a parse-tree style representation. The following exemplify the process. To find the variable which applies to a given component of the formula, just back up the tree until encountering an instance of that variable.

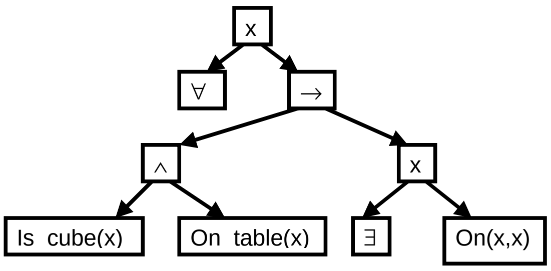
$(\forall x)((\text{Is\_cube}(x) \wedge \text{On\_table}(x)) \rightarrow (\exists y)\text{On}(y,x))$:



$(\forall x)(\exists y)((\text{Is\_cube}(x) \wedge \text{On\_table}(x)) \rightarrow \text{On}(y,x))$:



$(\forall x)((\text{Is\_cube}(x) \wedge \text{On\_table}(x)) \rightarrow (\exists x)\text{On}(x,x))$:



Free variables:

The formula

$$(\forall x)((\text{Is\_cube}(x) \wedge \text{On\_table}(x)) \rightarrow \text{On}(y,x))$$

presents quite a different situation from the previous three. The variable y is not bound by any quantifier at all. It is said to be a *free* variable.

The above formula cannot be regarded as a statement which is true or false unconditionally. Rather, it must be viewed as a parameterized statement, which is either true or false, depending upon the value of y. A request to supply the set of all values for y for which it is true is called a *query*, and is illustrated in the following.

$\{ y \mid (\forall x)((\text{Is\_cube}(x) \wedge \text{On\_table}(x)) \rightarrow \text{On}(y,x))\}$

This query identifies the set of all objects which are atop every block which is on the table. Of course, in this simple example, there can be at most one such y.

Notationally, $[\![\varphi]\!]$, is the relation defined by the formula φ. We will not give a completely formal definition, but the idea will be clear from an example.

This idea is the foundation of query languages in relational database systems, including the standard query language SQL. Here is an extremely simple example.

Employee			
Name	SSN	Salary	Department
Smith	123-45-6789	50000	Admin
Jones	987-65-4321	60000	Maintenance
Nordmann	000-11-2222	70000	Research
Levesque	333-44-5555	85000	Admin

There is a single relation named Employee, with four attributes. The associated instance has exactly four tuples.

Here is the query which requests the name and salary of everyone who works in the Admin department.

$\{ (x,y) \mid (\exists z)\mathrm{Employee}(x,z,y,\mathrm{Admin}) \}$

It is defined by the relation

$[\![(\exists z)\mathrm{Employee}(x,z,y,\mathrm{Admin})]\!]$.

Although a formal language known as the *relational tuple calculus* expresses queries in a form very similar to this, in practical use one uses SQL, which would express this query as follows.

```
Select    Name, Salary
From      Employee
Where     Department = "Admin"
```

Formulas and Sentences:

Definition: A *sentence* is a wff which has no free variables.

Example: The following three formulas are all sentences.

$(\forall x)((\mathrm{Is\_cube}(x) \wedge \mathrm{On\_table}(x)) \rightarrow (\exists y)\mathrm{On}(y,x))$

$(\forall x)(\exists y)((\mathrm{Is\_cube}(x) \wedge \mathrm{On\_table}(x)) \rightarrow \mathrm{On}(y,x))$

$(\forall x)((\mathrm{Is\_cube}(x) \wedge \mathrm{On\_table}(x)) \rightarrow (\exists x)\mathrm{On}(x,x))$

The following formulas are not sentences:

$(\forall x)((\mathrm{Is\_cube}(x) \wedge \mathrm{On\_table}(x)) \rightarrow \mathrm{On}(y,x))$

$\mathrm{On}(P1,x)$

- Sentences have truth values.
- Formulas which are not sentences define relations.

Distributivity and Quantifiers:

Q: Are the following two formulas equivalent?

$(\exists x)(\mathrm{On}(P1,x) \vee \mathrm{On}(P2,x))$

$(\exists x)\mathrm{On}(P1,x) \vee (\exists x)\mathrm{On}(P2,x))$

A: Yes.

Q: Are the following two formulas equivalent?

$(\forall x)(\mathrm{Is\_cube}(x) \vee \mathrm{Is\_pyramid}(x))$

$(\forall x)(\mathrm{Is\_cube}(x) \vee (\forall x)\mathrm{Is\_pyramid}(x)$

A: No.

Observation: $\exists$ distributes over $\vee$, but $\forall$ does not. □

Q: Are the following two formulas equivalent?

$(\exists x)(\mathrm{On\_table}(x) \wedge \mathrm{On}(P1,x))$

$(\exists x)\mathrm{On\_table}(x) \wedge (\exists x)\mathrm{On}(P1,x)$

A: No.

Q: Are the following two formulas equivalent?

$(\forall x)(\mathrm{On\_table}(x) \wedge \mathrm{Green}(x))$

$(\forall x)(\mathrm{On\_table}(x) \wedge (\forall x)\mathrm{Green}(x)$

A: Yes.

Observation: $\forall$ distributes over $\wedge$, but $\exists$ does not. □

Valuations:

We now turn to the question of how to assign a truth value to a wff φ, given an interpretation v.

- In propositional logic, this was quite simple; we just substituted, into the formula φ, the truth values for the propositions as defined by v, and expanded.
- In first-order logic, the situation is complicated somewhat, since:
  - Truth is defined in terms of a relation containing certain domain elements.
  - The formal arguments of relation symbols in formulas are terms.
- Thus, it is necessary to define an association between terms and domain elements. This is the role of the valuation.

Example: To determine the truth value of On(x,y) relative to a given interpretation J, it is necessary to associate domain elements with x and y.

Basically, the idea is to begin with a function which associates a "membership value" with each variable, and then use the interpretation J to extend this function to arbitrary terms.

Definition: Let $\mathsf{T} = (\mathsf{V}, \mathsf{K}, \mathsf{F})$ be a term algebra, let L be a first-order logic, and let J be an interpretation of L. A J-valuation of T is a function

$$w : \text{Terms}(T) \to \text{Dom}(J)$$

satisfying the following constraints.

(a) If $t \in \mathsf{V}$ is a variable, there are no constraints on w(t), other than it be a member of Dom(J).

(b) If t is a constant symbol k, the J-valuation on k is the domain value associated with k in the interpretation J. Formally, if $t = k \in \mathsf{K}$, then $w^J(t) = k^J$.

(c) If t is a general term of the form $f(t_1, t_2, .., t_n)$, then the J-valuation on t is defined by applying this definition recursively to the components. Formally, if $t \in \text{Terms}(T)$ is of the form $f(t_1, t_2, .., t_n)$, then
$$w^J(t) = f^J(w^J(t_1), w^J(t_2) .., w^J(t_n)).$$

Tarskian Semantics:

We are finally in a position to define the semantics of wff's in first-order logic. The following formulation is called Tarskian semantics, in honor of its formulator, the logician Alfred Tarski.

Definition: Let L be a first-order logic, let J be an interpretation of L, and let w be a J-valuation for L. The truth function

$$\bar{w}: \text{SWF}(L) \to \{0,1\}$$

is defined as follows.

(a) $\bar{w}(\perp) = 0$.

(b) For a wff φ of the form $R(t_1, t_2, .., t_n)$,

$$\bar{w}(\varphi) = \begin{cases} 1 \text{ if } R(w^J(t_1), w^J(t_2), .., w^J(t_n)) \in R^J \\ 0 \text{ if } R(w^J(t_1), w^J(t_2), .., w^J(t_n)) \notin R^J \end{cases}$$

(c) $\bar{w}((\neg\varphi)) = 1 - \bar{w}(\neg\varphi)$

(d) $\bar{w}((\varphi_1 \to \varphi_2)) = \max(1 - \bar{w}(\varphi_1), \bar{w}(\varphi_2))$.

(e) $\bar{w}((\forall x)\varphi) = 1$ if $\bar{v}(\varphi) = 1$ for every J-valuation v with $v(y) = w(y)$ for every $y \in \mathsf{V} \setminus \{x\}$.
In other words, φ must be true whenever we use a valuation which differs from w only in the way in which it assigns a value to x. That is, it must be true for all assignments to x.

Theorem: Let J be an interpretation of the first-order logic L, and let $\varphi \in \text{SWF}(\mathsf{L})$. If φ is a sentence, then one of the following is true:

(a) For every J-valuation w of L, $\bar{w}(\varphi) = 1$.

(b) For every J-valuation w of L, $\bar{w}(\varphi) = 0$.

Proof: Let u and w each be J-evaluations of L, and let X be the set of variables which occur in L. Note that X must be finite. For any variable $x \in \mathsf{V} \setminus X$, since x does not occur in φ, neither $\bar{u}(\varphi)$ nor $\bar{w}(\varphi)$ can depend upon x. Therefore, without loss of generality, it may be assumed that u and w agree on all variables $x \in \mathsf{V} \setminus X$. Since X is a finite set, there must be J-valuations $w_1, w_2, .., w_k$, with the property that $w_1 = u$, $w_k = w$, and w_i differs from w_{i+1} on only one element of X. But then, in view of part (e) of the preceding definition, it must be that $\bar{w}_i(\varphi) = \bar{w}_j(\varphi)$ for all i, j. Thus, $\bar{u}(\varphi) = \bar{w}(\varphi)$, as was to be shown. □

The following parallels the propositional case:

Definition: Let J be an interpretation of the first-order logic L, and let $\varphi \in \text{SWF}(\mathsf{L})$ be a sentence.

(a) J is a *model* of φ if, for every J-evaluation w, $\bar{w}(\varphi) = 1$.

(b) The sentence φ is satisfiable if it has a model.

(c) The sentence φ is a tautology if every interpretation is a model.

(d) The symbol $\models$ has the same meaning as in propositional logic.

The following extensions, except for (e), are identical to those of propositional logic.

Facts: Let J be an interpretation of the first-order logic L, and let w be a J-valuation of L. Extend $\bar{w}: \text{WF}(\mathsf{L}) \to \{0,1\}$ as follows.

(a) $\bar{w}((\varphi_1 \vee \varphi_2)) = \max(\bar{w}(\varphi_1), \bar{w}(\varphi_2))$.

(b) $\bar{w}((\varphi_1 \wedge \varphi_2)) = \min(\bar{w}(\varphi_1), \bar{w}(\varphi_2))$.

(c) $\bar{w}((\varphi_1 \equiv \varphi_2)) = 1 - \text{abs}(\bar{w}(\varphi_1) - \bar{w}(\varphi_2))$.

(d) $\bar{w}(\top) = 1$.

(e) $\bar{w}((\exists x)\varphi) = 1$ if $\bar{v}(\varphi) = 1$ for some J-valuation v with $v(y) = w(y)$ for every $y \in \mathsf{V} \setminus \{x\}$. □

Decidability:

Contrary to the case of propositional logic, we have the following negative result.

Theorem: There is no algorithm which takes as inputs an arbitrary sentences in a first-order logic, and decides whether or not that sentence is a tautology. In other words, the question of whether or not

$$\models \varphi$$

holds, for an arbitrary sentence in first-order logic, is undecidable. □

This statement is equivalent to the unsolvability of the halting problem (for Turing machines), which you may have studied in other courses.

Corollary: The question of whether an arbitrary sentence in first-order logic is satisfiable is undecidable. □

Later, we will see that some special cases are decidable.