The semantics of predicate logic

Readings: Section 2.4, 2.5, 2.6.

In this module, we will precisely define the semantic interpretation of formulas in our predicate logic. In propositional logic, every formula had a fixed, finite number of models (interpretations); this is not the case in predicate logic. As a consequence, we must take more care in defining notions such as satisfiability and validity, and we will see that there cannot be algorithms to decide if these properties hold or not for a given formula.

Models (2.4.1)

In the semantics of propositional logic, we assigned a truth value to each atom. In predicate logic, the smallest unit to which we can assign a truth value is a predicate $P(t_1, t_2, \ldots, t_n)$ applied to terms.

But we cannot arbitrarily assign a truth value, as we did for propositional atoms. There needs to be some consistency.

We need to assign values to variables in appropriate contexts, and meanings to functions and predicates. Intuitively, this is straightforward, but we must define such things precisely in order to ensure consistency of interpretation.

Example

In Module 5, we considered the formula

$$\forall x(P(x) \wedge \neg Q(x) \rightarrow R(x)) \ .$$

Our interpretation of this statement was, “Every student who took CS245, but did not pass CS245, failed CS245.”

Under this interpretation, x ranges over all students (say, at UW).

So, since x is a placeholder for a term, terms t denote UW students.

P, Q, and R, then are properties of students. We can think of them as $\mathbb{B}$-valued functions on UW students:

$P(x) =$ “x took CS245”, $Q(x) =$ “x passed CS245”,
$R(x) =$ “x failed CS245”

More abstractly, P, Q, and R are sets:

$P = \{\text{students who took CS245}\}$

$Q = \{\text{students who passed CS245}\}$

$R = \{\text{students who failed CS245}\}$

Then $P(x)$ is shorthand for $x \in P$, and similarly for Q and R.

We could also, however, interpret the predicate symbols P, Q, and R as follows:

$P = \{\text{natural numbers}\}$

$Q = \{\text{even numbers}\}$

$R = \{\text{odd numbers}\}$

Then terms t range over some numeric domains, say the integers or the real numbers, and the formula says that within that domain, all natural numbers that are not even are odd.

As we see, there is no requirement that our interpretation be about UW students, regardless of our initial motivation!

Finally, we could also apply the following interpretation:

$P = \{\text{natural numbers}\}$

$Q = \{\text{even numbers}\}$

$R = \{\text{prime numbers}\}$

Then the formula says that all natural numbers that are not even are prime. This is clearly a false statement, but a possible interpretation of the formula.

Another example

Consider the following formula:

$$\forall x(\exists y L(x, y) \rightarrow L(x, c))$$

We can take our domain of concrete values, in this case, to include students and courses. Then c might denote the constant CS245, and L is a $\mathbb{B}$-valued function on two variables that we might define as follows:

$L(x, y) =$ "x is a student, y is a course, and x loves y"

Then the formula says that every student who loves a course must love CS245.

Again, we might rephrase L as a set, this time of ordered pairs:

$$L = \{(x, y) \mid x \text{ is a student, } y \text{ is a course, and } x \text{ loves } y\}$$

L is now a set of ordered pairs, which, in mathematical terms, we call a **relation**.

More generally, we call a set of ordered pairs a binary relation, a set of ordered triples a ternary relation, and a set of n-tuples an n-place (or n-ary) relation. Further, a set of singletons is a one-place (or unary) relation, or predicate. A zero-place (or nullary) relation corresponds to a nullary predicate, which we use to model atomic propositions, and is therefore either the constant T or the constant F.

Models

An interpretation of a logical formula, comprising semantics for terms, constants, function symbols, and predicate symbols, form what is called a **model**.

Note that each of our interpretations included the following elements:

- a domain of interpretation for values (e.g., UW Students, integers, real numbers, etc.);
- meanings for each n-ary function symbol as n-ary domain-valued functions on the the domain of interpretation;
- meanings for each n-ary predicate symbol as n-place relations.

These are the essential components of a model.

Here is the precise definition of a model $\mathcal{M}$ for a given set of predicate symbols $\mathcal{P}$ and function symbols $\mathcal{F}$:

1. A nonempty set $A^{\mathcal{M}}$ (the universe of concrete values);
2. A concrete element $f^{\mathcal{M}}$ of $A^{\mathcal{M}}$ for every nullary function symbol (constant) $f \in \mathcal{F}$;
3. A concrete function $f^{\mathcal{M}} : (A^{\mathcal{M}})^n \to A^{\mathcal{M}}$ for every n-ary function symbol $f \in \mathcal{F}$;
4. A subset $P^{\mathcal{M}}$ of n-tuples over $A^{\mathcal{M}}$ (i.e., an n-place relation on $A^{\mathcal{M}}$) for every n-ary predicate symbol $P \in \mathcal{P}$.

The textbook leaves off the superscript $\mathcal{M}$ on $A^{\mathcal{M}}$, which is unambiguous if the model is clear.

Interpreting terms in a model

Now that we have formalized the domains in which our interpretations of predicate logic formulas will reside—models—we can discuss how to interpret terms and predicates within a model.

Let $\mathcal{M}$ be a model and t be a term without variables. Then $t^{\mathcal{M}}$, the interpretation of t in $\mathcal{M}$, is given as follows:

- if t is a constant c, then $t^{\mathcal{M}} = c^{\mathcal{M}}$;
- if $t = f(t_1, \ldots, t_n)$, where f is an n-ary function symbol, then $t^{\mathcal{M}} = f^{\mathcal{M}}(t_1^{\mathcal{M}}, \ldots, t_n^{\mathcal{M}})$.

Note that, strictly speaking, the second clause subsumes the first, but we keep them separate for the sake of clarity.

Interpreting predicates in a model

Interpreting predicates in a model is similar to interpreting terms.

Let $\mathcal{M}$ be a model, P be a predicate symbol of arity n, and $t_1, \ldots, t_n$ variable-free terms. Then we define

$$(P(t_1, \ldots, t_n))^{\mathcal{M}} = P^{\mathcal{M}}(t_1^{\mathcal{M}}, \ldots, t_n^{\mathcal{M}}) \, .$$

But the model is the easy part....

What do we do about variables?

Suppose we have a formula such as

$$\forall x(\exists y L(x, y) \rightarrow L(x, c)) \ ,$$

where c is a constant (or nullary function symbol). Given a model $\mathcal{M}$, how do we interpret a predicate like $L(x, c)$ in $\mathcal{M}$? (We'll ignore the quantifiers for the moment, and come back to them later.)

According to the definition, it would be something like

$$L^{\mathcal{M}}(x^{\mathcal{M}}, c^{\mathcal{M}}) \ .$$

But x is a variable (note that x is free in $L(x, c)$), so how can we evaluate $x^{\mathcal{M}}$? We can't find a specific element of the domain to assign to x, because we don't know what x is supposed to be!

The expression $x^{\mathcal{M}}$ does not make sense.

Variables and environments

We need a way to intepret variables in a model.

Key observation: the meaning of a variable depends on (**is parameterized by**) the specific value of the domain that gets assigned to the variable.

Can we capture the notion of assigning values to variables as a mathematical object?

Yes—we call it an **environment**.

An environment is basically a list of variable names and elements of the domain to be assigned to them.

Example: if $A^{\mathcal{M}} = \{a, b, c\}$, then an environment σ might be $\sigma = \{(x, a), (y, a), (z, b)\}$, where x, y, and z are variables.

Hence, an environment is essentially a look-up table between variables and domain elements.

The **domain** of an environment is the set of variables upon which it operates. In our example, the domain of σ, denoted dom σ, is the set $\{x, y, z\}$.

As the terminology suggests, we can view an environment as a function that maps variables to domain elements. Hence, given a variable v and an environment σ, we denote by $\sigma(v)$ the result of looking up v in σ.

In our example above, $\sigma(x) = a$, $\sigma(y) = a$, and $\sigma(z) = b$.

We extend environments as follows: if σ is an environment and $c \in A^{\mathcal{M}}$, then we denote by $\sigma[x \mapsto c]$ the environment σ' that is identical to σ, except that $\sigma'(x) = c$.

Example: if $A^{\mathcal{M}} = \{a, b, c\}$ and $\sigma = \{(x, a), (y, c)\}$, then $\sigma[z \mapsto b] = \{(x, a), (y, c), (z, b)\}$.

Further,

$$\begin{aligned} \sigma[z \mapsto b][z \mapsto a] &= \{(x, a), (y, c), (z, b)\}[z \mapsto a] \\ &= \{(x, a), (y, c), (z, a)\} \ . \end{aligned}$$

Environments give us a way to interpret terms that contain variables.

Since the meaning of a variable (hence of the term that contains it) is dependent on the value assigned to it (which is encoded in an environment), we can interpret a term t as a **function** parameterized by an environment σ, such that dom σ contains the (free) variables of t.

Example: Suppose $t = g(f(x), c)$. We can interpret $t^{\mathcal{M}}$ as a function on environments σ whose domain includes x, as follows:

$$\begin{aligned} t^{\mathcal{M}}(\sigma) &= g^{\mathcal{M}}(f(x)^{\mathcal{M}}(\sigma), c^{\mathcal{M}}(\sigma)) \\ &= g^{\mathcal{M}}(f^{\mathcal{M}}(\sigma(x)), c^{\mathcal{M}}) \,. \end{aligned}$$

Note that constants (like c) are also interpreted as functions on environments; they just don't do anything with the environments.

Notice how the interpretation of t interprets the variable x as whatever it is assigned to by σ, the environment by which $t^{\mathcal{M}}$ itself is parameterized.

The same interpretation technique applies to predicates. We now interpret a predicate application $P(t_1, \ldots, t_n)$ as a $\mathbb{B}$-valued function on an environment σ whose domain includes the free variables of $t_1, \ldots, t_n$.

Example: Consider our earlier example, $L(x, c)$. Then we have

$$(L(x, c))^{\mathcal{M}}(\sigma) = L^{\mathcal{M}}(\sigma(x), c^{\mathcal{M}}) \ .$$

What we have so far

We would like to have a notion of an interpretation of formulas in predicate logic, analogous to our interpretation functions Φ in propositional logic.

For predicate logic, interpretation functions must be taken in the context of a particular model $\mathcal{M}$. For a predicate formula ψ, therefore, we use $\Phi^{\mathcal{M}}$ to denote an interpretation within the model $\mathcal{M}$, and write the interpretation of ψ as

$$\Phi^{\mathcal{M}}(\psi) \ .$$

Note that $\Phi^{\mathcal{M}}(\psi)$ must itself be a function on environments whose domain includes the free (but not the bound) variables of ψ.

So far, we are able to express the interpretations of terms and predicate applications:

$$\Phi^{\mathcal{M}}(P(t_1, \ldots t_n))(\sigma) = P^{\mathcal{M}}(\Phi^{\mathcal{M}}(t_1)(\sigma), \ldots, \Phi^{\mathcal{M}}(t_n)(\sigma))$$

Note that the action of Φ is completely determined by the model $\mathcal{M}$, so there is no difference between

$$\Phi^{\mathcal{M}}(P(t_1, \ldots t_n))$$

and

$$(P(t_1, \ldots t_n))^{\mathcal{M}} \ .$$

We use the $\Phi^{\mathcal{M}}$ notation primarily to enforce the analogy with the semantics of propositional logic.

What comes next

We still need to extend the definition of our interpretation function $\Phi^{\mathcal{M}}$ to accommodate connectives and quantifiers.

Interpreting connectives

In the context of propositional logic, we defined

$$\Phi(\psi_1 \Box \psi_2) = \mathrm{meaning}(\Box)(\Phi(\psi_1), \Phi(\psi_2)) \; .$$

For predicate logic, the definition is much the same, except that we must also account for the additional environment parameter:

$$\Phi^{\mathcal{M}}(\psi_1 \Box \psi_2)(\sigma) = \mathrm{meaning}(\Box)(\Phi^{\mathcal{M}}(\psi_1)(\sigma), \Phi^{\mathcal{M}}(\psi_2)(\sigma)) \; .$$

The definition of the $\mathrm{meaning}$ function for each connective is unchanged.

For the unary negation operator, $\neg$, we have, of course,

$$\Phi^{\mathcal{M}}(\neg\psi)(\sigma) = \mathrm{meaning}(\neg)(\Phi^{\mathcal{M}}(\psi)(\sigma)) \; .$$

Interpreting quantifiers

Finding definitions for $\Phi^{\mathcal{M}}(\forall x \psi)$ and $\Phi^{\mathcal{M}}(\exists x \psi)$ is trickier. We will have to reason carefully about functions on environments.

To begin, we will define two special constant functions $T^{\mathcal{M}}$ and $F^{\mathcal{M}}$:

$$T^{\mathcal{M}}(c) := T$$

$$F^{\mathcal{M}}(c) := F$$

These are the constant $\mathbb{B}$-valued functions for truth and falsehood that take an arbitrary $c \in A^{\mathcal{M}}$, ignore it, and simply return T and F, respectively.

Consider the expression $\Phi^{\mathcal{M}}(\forall x \psi)$. This interpretation denotes a $\mathbb{B}$-valued function on environments σ such that dom σ contains the free (but not the bound) variables in $\forall x \psi$.

In particular, dom σ does **not** contain x.

So let σ_0 be an environment that contains the free (but not the bound) variables in $\forall x \psi$, so that

$$\Phi^{\mathcal{M}}(\forall x \psi)(\sigma_0)$$

is a valid $\mathbb{B}$-valued expression. Then since $x \notin$ dom σ_0, the expression

$$\Phi^{\mathcal{M}}(\psi)(\sigma_0)$$

is **not** valid, because x is (presumably) free in ψ.

Hence, we cannot move directly from $\Phi^{\mathcal{M}}(\forall x \psi)(\sigma_0)$ to some expression involving $\Phi^{\mathcal{M}}(\psi)(\sigma_0)$.

In order to apply $\Phi^{\mathcal{M}}$ to ψ, we must extend the environment σ_0 to include x.

Suppose we wish to assign x the value $c \in A^{\mathcal{M}}$. Then we would interpret ψ in the environment $\sigma_0[x \mapsto c]$:

$$\Phi^{\mathcal{M}}(\psi)(\sigma_0[x \mapsto c])$$

Since we don't know which c we want to assign to x, what we are really describing is a $\mathbb{B}$-valued function on $A^{\mathcal{M}}$:

$$\text{the function } f \text{ such that } f(c) = \Phi^{\mathcal{M}}(\psi)(\sigma_0[x \mapsto c])$$

Aside: λ-notation

Creating functions using notation like

$$\text{the function } f \text{ such that } f(c) = \Phi^{\mathcal{M}}(\psi)(\sigma_0[x \mapsto c])$$

is really cumbersome. The problem is the need, within our notation, for every function to have a name. To streamline things a bit, we will introduce the notation

$$\lambda c.\Phi^{\mathcal{M}}(\psi)(\sigma_0[x \mapsto c])$$

to mean

$$\text{the function } f \text{ such that } f(c) = \Phi^{\mathcal{M}}(\psi)(\sigma_0[x \mapsto c]) \ .$$

In general, the expression

$$\lambda x.E$$

denotes a function that takes a parameter x and returns the value E.

Observe that our constant functions $T^{\mathcal{M}}$ and $F^{\mathcal{M}}$ can also be expressed in λ-notation:

$$T^{\mathcal{M}} = \lambda c.T$$

$$F^{\mathcal{M}} = \lambda c.F$$

There is a rich body of theory associated with the λ-notation (called the λ**-calculus**), but we do not discuss it here (see CS442). We use the notation simply as a convenience.

Back to interpreting quantifiers...

So far, we have broken down $\Phi^{\mathcal{M}}(\forall x\psi)(\sigma_0)$ into the function

$$\lambda c.\Phi^{\mathcal{M}}(\psi)(\sigma_0[x \mapsto c])$$

that takes as a parameter the value we wish to assign to x and returns an interpretation of the formula (i.e., a truth value).

Now, since we are interpreting the universally quantified formula $\forall x\psi$, we would like the interpretation to return T **independent of our choice of** c.

Thus, we interpret $\Phi^{\mathcal{M}}(\psi)(\sigma)$ as follows:

$$\Phi^{\mathcal{M}}(\forall x\psi)(\sigma) = \begin{cases} T & \text{if } (\lambda c.\Phi^{\mathcal{M}}(\psi)(\sigma[x \mapsto c])) = T^{\mathcal{M}} \\ F & \text{otherwise} \end{cases}$$

(σ_0 renamed back to σ for brevity)

For existential quantifiers, the development is similar. We now wish to define $\Phi^{\mathcal{M}}(\exists x\psi)(\sigma)$. As before, we arrive at the function

$$\lambda c.\Phi^{\mathcal{M}}(\psi)(\sigma_0[x \mapsto c]) \ .$$

This time, however, we do not need this function to return T for all values of c, but only for at least one value of c. Equivalently, we want that the function will **not** return F for all values of c. Hence, we can interpret the existential formula $\exists x\psi$ as follows:

$$\Phi^{\mathcal{M}}(\exists x\psi)(\sigma) = \begin{cases} F & \text{if } (\lambda c.\Phi^{\mathcal{M}}(\psi)(\sigma[x \mapsto c])) = F^{\mathcal{M}} \\ T & \text{otherwise} \end{cases}$$

In summary, the interpretations of quantified formulas are as follows:

$$\Phi^{\mathcal{M}}(\forall x\psi)(\sigma) = \begin{cases} T & \text{if } (\lambda c.\Phi^{\mathcal{M}}(\psi)(\sigma[x \mapsto c])) = T^{\mathcal{M}} \\ F & \text{otherwise} \end{cases}$$

$$\Phi^{\mathcal{M}}(\exists x\psi)(\sigma) = \begin{cases} F & \text{if } (\lambda c.\Phi^{\mathcal{M}}(\psi)(\sigma[x \mapsto c])) = F^{\mathcal{M}} \\ T & \text{otherwise} \end{cases}$$

For the sake of comparison, here are the (slightly rephrased) interpretations of conjunction and disjunction in propositional logic:

$$\Phi(\psi_1 \wedge \psi_2) = \begin{cases} T & \text{if } \Phi(\psi_i) = T \text{ for each } i \\ F & \text{otherwise} \end{cases}$$

$$\Phi(\psi_1 \vee \psi_2) = \begin{cases} F & \text{if } \Phi(\psi_i) = F \text{ for each } i \\ T & \text{otherwise} \end{cases}$$

The interpretations for quantifiers are more complex, but clearly inspired by, the interpretations for $\wedge$ and $\vee$.

As an aside, the dependence of interpretations on environments can also be expressed using λ-notation:

$$\Phi^{\mathcal{M}}(\forall x\psi) = \lambda\sigma. \begin{cases} T & \text{if } (\lambda c.\Phi^{\mathcal{M}}(\psi)(\sigma[x \mapsto c])) = T^{\mathcal{M}} \\ F & \text{otherwise} \end{cases}$$

$$\Phi^{\mathcal{M}}(\exists x\psi) = \lambda\sigma. \begin{cases} F & \text{if } (\lambda c.\Phi^{\mathcal{M}}(\psi)(\sigma[x \mapsto c])) = F^{\mathcal{M}} \\ T & \text{otherwise} \end{cases}$$

Semantic entailment in predicate logic

We are now in a position do define a semantic entailment in the context of predicate logic.

We say $\phi_1, \ldots, \phi_n \models \psi$ (i.e., $\phi_1, \ldots, \phi_n$ **entail** ψ) if for all models $\mathcal{M}$ and all environments σ, if

$$\Phi^{\mathcal{M}}(\phi_1)(\sigma) = \cdots = \Phi^{\mathcal{M}}(\phi_n)(\sigma) = T\,,$$

then

$$\Phi^{\mathcal{M}}(\psi)(\sigma) = T\,.$$

Notice that this definition requires that the entailment hold **for all models**—of which there are infinitely many; hence, simple truth table do not suffice to establish semantic truths in predicate logic!

Time for an example

Consider the sentence “None of Alma’s lovers’ lovers love her.” Here “Alma” can be a constant a, and the concept “ x loves y” can be a binary predicate $L(x, y)$. We can then represent the above sentence as the following formula:

$$\forall x \forall y (L(x, a) \wedge L(y, x) \rightarrow \neg L(y, a)).$$

Intuitively, $L(x, a)$ says that x is Alma’s lover, and $L(y, x)$ says that y loves x, so $L(x, a) \wedge L(y, x)$ says that y is one of Alma’s lovers’ lovers. $\neg L(y, a)$ says that y does not love Alma, and the quantifiers make sure this is true for any x, y.

Consider the model $\mathcal{M}$ defined by $A^{\mathcal{M}} = \{p, q, r\}$, $a^{\mathcal{M}} = p$, $L^{\mathcal{M}} = \{(p, p), (q, p), (r, p)\}$. How can we decide if $\mathcal{M}$ models this formula?

One way to do it is to apply the definition of $\Phi^{\mathcal{M}}$ systematically—but this is cumbersome in the presence of quantifiers.

Instead, let us focus on the subformulas $L(x,a) \wedge L(y,x)$ and $\neg L(y,a)$.

In order to interpret these formulas, we need an environment. Let σ be the environment $\{(x,p),(y,q)\}$. Then

$$\begin{aligned}
& \Phi^{\mathcal{M}}(L(x,a) \wedge L(y,x))(\sigma) \\
= \quad & \mathrm{meaning}(\wedge)(\Phi^{\mathcal{M}}(L(x,a))(\sigma), \Phi^{\mathcal{M}}(L(y,x))(\sigma)) \\
= \quad & \mathrm{meaning}(\wedge)(L^{\mathcal{M}}(\sigma(x), a^{\mathcal{M}}), L^{\mathcal{M}}(\sigma(y), \sigma(x))) \\
= \quad & \mathrm{meaning}(\wedge)(L^{\mathcal{M}}(p,p), L^{\mathcal{M}}(q,p)) \\
= \quad & \mathrm{meaning}(\wedge)(T,T) \\
= \quad & T
\end{aligned}$$

For $\neg L(y, a)$, we have

$$\begin{aligned}
\Phi^{\mathcal{M}}(\neg L(y,a))(\sigma) &= \text{meaning}(\neg)(\Phi^{\mathcal{M}}(L(y,a))(\sigma)) \\
&= \text{meaning}(\neg)(L^{\mathcal{M}}(\sigma(y), a^{\mathcal{M}})) \\
&= \text{meaning}(\neg)(L^{\mathcal{M}}(q,p)) \\
&= \text{meaning}(\neg)(T) \\
&= F
\end{aligned}$$

Putting these together, we have

$$\begin{aligned}
& \Phi^{\mathcal{M}}(L(x,a) \wedge L(y,x) \rightarrow \neg L(y,a))(\sigma) \\
= \;& \text{meaning}(\rightarrow)(\Phi^{\mathcal{M}}(L(x,a) \wedge L(y,x))(\sigma), \Phi^{\mathcal{M}}(\neg L(y,a))(\sigma)) \\
= \;& \text{meaning}(\rightarrow)(T, F) \\
= \;& F
\end{aligned}$$

Now, let σ_0 be an environment whose domain does not include x and y. Then we have

$$\begin{aligned} & \Phi^{\mathcal{M}}(L(x,a) \wedge L(y,x) \rightarrow \neg L(y,a))(\sigma) \\ = \quad & \Phi^{\mathcal{M}}(L(x,a) \wedge L(y,x) \rightarrow \neg L(y,a))(\sigma_0[x \mapsto p][y \mapsto q]) \\ = \quad & (\lambda b.\lambda c.\Phi^{\mathcal{M}}(L(x,a) \wedge L(y,x) \rightarrow \neg L(y,a))(\sigma_0[x \mapsto b][y \mapsto c]))pq \end{aligned}$$

Since $\Phi^{\mathcal{M}}(L(x,a) \wedge L(y,x) \rightarrow \neg L(y,a))(\sigma) = F$, the expression

$$\lambda c.\Phi^{\mathcal{M}}(L(x,a) \wedge L(y,x) \rightarrow \neg L(y,a))(\sigma_0[x \mapsto a][y \mapsto c])$$

cannot be equal to $T^{\mathcal{M}}$. Hence,

$$\Phi^{\mathcal{M}}(\forall y(L(x,a) \wedge L(y,x) \rightarrow \neg L(y,a))(\sigma[x \mapsto p]) = F$$

But then

$$\lambda b.\Phi^{\mathcal{M}}(\forall y(L(x,a) \wedge L(y,x) \rightarrow \neg L(y,a))(\sigma[x \mapsto b])$$

cannot be equal to $T^{\mathcal{M}}$. Hence,

$$\Phi^{\mathcal{M}}(\forall x \forall y(L(x,a) \wedge L(y,x) \rightarrow \neg L(y,a))(\sigma) = F$$

and the formula is false under this model. Hence, it is not a semantic truth.

We have treated this example very formally. Informally, we can reason about the formula in a much more intuitive way. Since the statement has the form $\forall x \forall y \phi$, refuting it simply amounts to finding specific assignments to x and y such that $\neg\phi$ under these assignments.

Quantifier rules ... informally

In order to facilitate semantic reasoning about quantified formulas, we will take a moment to give a more intuitive characterization for them.

Formally, $\forall x \phi$ is interpreted as follows:

$$\Phi^{\mathcal{M}}(\forall x \psi)(\sigma) = \begin{cases} T & \text{if } (\lambda c. \Phi^{\mathcal{M}}(\psi)(\sigma[x \mapsto c])) = T^{\mathcal{M}} \\ F & \text{otherwise} \end{cases}$$

We can rewrite the top condition as follows:

$$\Phi^{\mathcal{M}}(\forall x \psi)(\sigma) = \begin{cases} T & \text{if } \Phi^{\mathcal{M}}(\psi)(\sigma[x \mapsto c])) = T \text{ for every } c \\ F & \text{otherwise} \end{cases}$$

Thus $\Phi^{\mathcal{M}}(\forall x \psi)(\sigma) = T$ if every assignment of a value to x in ψ yields semantic truth.

Similarly, $\exists x\phi$ is interpreted as follows:

$$\Phi^{\mathcal{M}}(\forall x\psi)(\sigma) = \begin{cases} F & \text{if } (\lambda c.\Phi^{\mathcal{M}}(\psi)(\sigma[x \mapsto c])) = F^{\mathcal{M}} \\ T & \text{otherwise} \end{cases}$$

We can rewrite the top condition as follows:

$$\Phi^{\mathcal{M}}(\forall x\psi)(\sigma) = \begin{cases} F & \text{if } \Phi^{\mathcal{M}}(\psi)(\sigma[x \mapsto c])) = F \text{ for every } c \\ T & \text{otherwise} \end{cases}$$

Thus $\Phi^{\mathcal{M}}(\exists x\psi)(\sigma) = T$ if some assignment of a value to x in ψ yields semantic truth.

In predicate logic, we say $\Gamma \models \psi$ if and only if for all models $\mathcal{M}$ and all environments σ, whenever $\Phi^{\mathcal{M}}(\phi)(\sigma) = T$ for all $\phi \in \Gamma$, then $\Phi^{\mathcal{M}}(\psi)(\sigma) = T$ as well.

This seems a very strong condition: how do we check this for all possible models? We will demonstrate that this is possible, through careful reasoning, for the quantifier equivalences discussed in the last module. But first we will define notions of satisfiability, validity, and consistency.

The formula ψ is valid if and only if $\models \psi$. Note that this condition is implicitly quantified universally over all models and all environments.

The set Γ is consistent (or satisfiable) if and only if there is a model $\mathcal{M}$ and an environment σ such that for all $\phi \in \Gamma$, $\Phi^{\mathcal{M}}(\phi)(\sigma) = T$.

In the previous module, we proved $\forall x(P(x) \vee Q(x)), \exists x(\neg P(x)) \vdash \exists x Q(x)$. Now we will give an argument that $\forall x(P(x) \vee Q(x)), \exists x(\neg P(x)) \models \exists x Q(x)$.

Consider a model $\mathcal{M}$ satisfying $\forall x(P(x) \vee Q(x))$ and $\exists x(\neg P(x))$. The truth of the second formula tells us that there is an $a \in A^{\mathcal{M}}$ such that $(a) \notin P^{\mathcal{M}}$. But the truth of the first formula tells us that for an environment σ that maps x to a, $P(x) \vee Q(x)$ is assigned true. Since $P(x)$ isn't true in σ, $Q(x)$ must be. And with $Q(x)$ true in some environment, $\exists x Q(x)$ is true in $\mathcal{M}$, which shows that the original entailment holds.

The informal argument on the previous slide avoided much notation; here is a more precise though perhaps less readable version.

Suppose $\Phi^{\mathcal{M}}(\forall x(P(x) \vee Q(x)))(\sigma) = T$ and $\Phi^{\mathcal{M}}(\neg P(x))(\sigma) = T$.

Then for some $a \in A^{\mathcal{M}}$, $\Phi^{\mathcal{M}}(\neg P(x))(\sigma[x \mapsto a]) = T$, and so $\Phi^{\mathcal{M}}(P(x))(\sigma[x \mapsto a]) = F$. On the other hand, $\Phi^{\mathcal{M}}(\forall x(P(x) \vee Q(x)))(\sigma) = T$ means that $\Phi^{\mathcal{M}}(P(x) \vee Q(x))(\sigma[x \mapsto a]) = T$. Knowing that $\Phi^{\mathcal{M}}(P(x))(\sigma[x \mapsto a]) = F$, we must have $\Phi^{\mathcal{M}}(Q(x))(\sigma[x \mapsto a]) = T$, and thus $\Phi^{\mathcal{M}}(\exists x Q(x))(\sigma)$.

The semantics of equality (2.4.3)

We use the term **intensional equality** to mean equality in the syntactic sense, as in $t = t$ for any term t.

But we also want to talk about equality in a semantic sense. Suppose in a particular model $\mathcal{M}$, $f^{\mathcal{M}}$ maps the interpretation of a to c, and $g^{\mathcal{M}}$ maps the interpretation of b to c. We then want the formula $f(a) = g(b)$ to be assigned T.

We ensure this by mandating that $=^{\mathcal{M}}$ should always be the equality relation on the set $A^{\mathcal{M}}$. This notion is called **extensional equality**.

Soundness and completeness

Intuitively, the soundness of predicate logic, which means that $\Gamma \vdash \psi$ implies $\Gamma \models \psi$, is not surprising; the rules of natural deduction are set up to preserve truth under interpretation.

We can see how the earlier proof of the soundness of propositional logic, which proceeded by structural induction on formulas, can be extended to cover predicate logic. Formulas are a bit more complicated, and we need some arguments about models similar to the one we just did.

The completeness of predicate logic means that $\Gamma \models \psi$ implies $\Gamma \vdash \psi$. This is more surprising, because we cannot mimic our earlier proof for propositional logic. That put together information from 2^n valuations (models) of a formula to yield a long finite proof.

But for predicate logic, we may not have a finite number of models, and they may be of very different types. It is not at all clear how to put all this information together to yield a proof.

The completeness of predicate logic was proved by Kurt Gödel in his Ph.D dissertation for the University of Vienna in 1930. A number of simpler proofs have been given by others, notably Henkin and Herbrand.

Gödel is more famous for two incompleteness theorems, which we will discuss shortly. Both the completeness and incompleteness theorems are proved in detail in PMath 432.

Rather than give the proofs of soundness and completeness, we will discuss some implications of them.

The most immediate implication is that, as with propositional logic, we have a way to show that a formula ϕ does not have a proof in natural deduction. By soundness, it suffices to demonstrate a model in which it is assigned false by our semantics.

This corresponds to the practice of demonstrating that a claim is false by showing a counterexample.

As an example of the use of counterexamples to demonstrate invalidity, consider the sequent $\forall x(P(x) \vee Q(x)) \vdash \forall x P(x) \vee \forall x Q(x)$. We will show that this is invalid by giving a model which satisfies the LHS but not the RHS.

Let $A = \{a, b\}$, $P^{\mathcal{M}} = \{(a)\}$, $Q^{\mathcal{M}} = \{(b)\}$, and let σ denote the empty environment. Then $\Phi^{\mathcal{M}}(\forall x(P(x) \vee Q(x)))(\sigma) = T$, but $\Phi^{\mathcal{M}}(\forall x P(x) \vee \forall x Q(x))(\sigma) = F$.

Thus $\forall x(P(x) \vee Q(x)) \not\models \forall x P(x) \vee \forall x Q(x)$, and by soundness, $\forall x(P(x) \vee Q(x)) \nvdash \forall x P(x) \vee \forall x Q(x)$.

Gödel's proof of completeness was not **effective**; it did not provide a method for definitely deciding whether a formula was provable or not. Note that we have such a method for propositional logic; we can simply try all 2^n possible interpretations, where n is the number of atoms in the formula.

Since a proof can be mechanically checked for validity, we can obtain a partial result; if a formula is provable, we can find a proof by generating all possible strings over the alphabet in which our proofs are written, and testing each one to see if it is a valid proof of the formula. This is not necessarily a fast or efficient algorithm, but it will find the proof. Provability is **semi-decidable**.

There does not seem to be any corresponding way to conclude that no proof exists.

By completeness, if we can find a model in which the formula is not valid, we can conclude that it is not provable. But such a model may be infinite, and we cannot check all possible interpretations of functions and predicates. In fact, there is no algorithm to test whether a formula in predicate logic is provable or not; this is **undecidable**.

To show this requires a formal model of computation. Such a model of computation, and a proof of the undecidability of provability, was first described by the American logician Alonzo Church in 1936. His model of computation was the lambda calculus, and his proof made heavy use of Gödel's incompleteness result.

Independently, a few months later, the British mathematician Alan Turing came up with a different model and a simpler proof.

Both Church's and Turing's proofs first demonstrated that there was no algorithm to answer questions about programs. Church showed it was impossible to decide whether two programs were functionally identical, and Turing showed that it was impossible to decide whether a program would halt.

They then showed that predicate logic could express the questions they were asking; hence it is also undecidable. A similar proof is given in section 2.5 of the text, where predicate logic is used to express the existence of a solution to a "correspondence problem" defined by the American mathematician Emil Post.

Post's correspondence problem is proved undecidable in CS 365; Turing's proof is given in CS 365 and 360; and Church's lambda calculus is studied in CS 442.

These results, proved in the 1930's, demonstrated limitations to computation even before electronic computers existed, and were a powerful influence on the subsequent development of hardware and software.

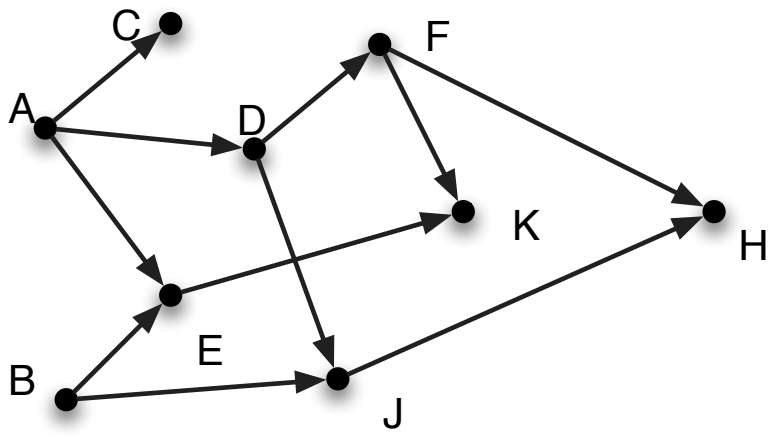
They continue to be important. We saw earlier that while satisfiability, validity, and provability are all decidable for propositional logic, they apparently cannot be determined efficiently for general formulas. Now we see that the greater expressivity of predicate logic makes these questions undecidable.

The tradeoff between expressivity and efficiency, in both theoretical and practical terms, is the subject of active research.

Expressiveness of predicate logic (2.6)

Because many properties of our specifications, our algorithms, and our programs can be expressed in logical terms, we naturally wish to make our logical languages as expressive as possible, though not at the cost of being unable to work with the resulting formulas.

Our example of the limited expressibility of first-order predicate logic will involve directed graphs, which are used in CS 135, CS 240, Math 239, CS 341, and many other courses. A directed graph G consists of a finite set V of vertices and an binary edge relation E. We often visualize graphs by drawing dots to represent each of the elements of V, and drawing an arrow from x to y iff $(x, y) \in E$ (in which case we say "there is an edge from x to y").



To represent graphs in the language of predicate logic, we let the variables stand for nodes (vertices), and use a binary predicate symbol E for the edge relation. That is, if a graph G contains an edge from u to v, then $E(u, v)$ is true in the model defined by G.

As an example, the formula $\exists x \exists y E(x, y)$ is true for graphs that have at least one edge. This is the formalization of “The graph has at least one edge.”

What formulas express the following statements about graphs?

“The graph has at least two vertices.”

“The graph has at least one edge going from a vertex to itself.”

“The graph contains a vertex to which no edge goes.”

Going in the other direction, how can we intuitively express the following formulas?

$\exists x \forall y E(y, x)$

$\forall x \forall y (E(x, y) \rightarrow E(y, x))$

$\forall x \forall y E(x, y)$

The question of reachability in directed graphs (given a graph G and two nodes u, v, is there a directed path from u to v?) is important in many areas of computer science. A program using pointer-based data structures is free of memory leaks if every allocated segment of memory can be reached from a program variable. Finding solutions to solitaire puzzles, or more generally, goal search and motion planning can be expressed in terms of reachability.

Math 239 and CS 341 cover efficient algorithms for reachability when the graph is given explicitly. But there are many computational situations where the graph is not explicit. For example, the nodes of the graph could be states of a program or system, and the edges could represent steps or transitions. Questions of freedom from error, safety, or freedom from deadlock can then be expressed in terms of reachability.

It is therefore surprising to learn that reachability cannot be expressed in predicate logic. Most of the “impossibility” results discussed in this course are only stated, not proved, but we can prove this one from completeness, by way of a couple of nice results in formal logic. First, though, we should discuss what it means to attempt to express reachability in predicate logic.

To talk about reachability using a formula, we again let the variables stand for nodes (vertices), and use binary predicate symbol E for the edge relation. That is, if in a particular graph G, there is an edge from u to v, then $E(u, v)$ is true in the model defined by G.

A formula describing a more general relationship between u and v is one in which u and v are the only free variables. For instance, the formula $\exists x(E(u, x) \wedge E(x, v))$ is made true by a model defined by G if and only if there is a path of length 2 in G.

The difficulty comes because a path from u to v in an arbitrary graph can have unbounded (though finite) length, and we need to express reachability with a fixed (finite) formula.

We begin our proof that reachability is not expressible in predicate logic with an important and general result.

Compactness Theorem (2.24): Let Γ be a (possibly infinite) set of sentences of predicate logic. If all finite subsets of Γ are satisfiable, then so is Γ.

Proof: Suppose that all finite subsets of Γ are satisfiable, but Γ is not satisfiable. Then $\Gamma \models \bot$ (since no model makes all $\phi \in \Gamma$ true). By completeness, $\Gamma \vdash \bot$. This must have a finite proof, mentioning only a finite subset Δ of sentences from Γ. Then $\Delta \vdash \bot$, and by soundness, $\Delta \models \bot$. But this is a contradiction to the assumption that Δ, as a finite subset of Γ, is satisfiable. $\square$

As a warmup on the use of compactness, we prove one of a number of related theorems with the names of Löwenheim and Skolem on them.

Theorem (2.25): Let ψ be a sentence of predicate logic such that for any natural number $n \geq 1$, there is a model of ψ with at least n elements. Then ψ has a model with infinitely many elements.

Proof: For all n, let ϕ_n be defined as:

$$\exists x_1 \exists x_2 \ldots \exists x_n \bigwedge_{1 \leq i < j \leq n} \neg(x_i = x_j).$$

ϕ_n asserts that there are at least n elements. Now define $\Gamma = \{\psi\} \cup \{\phi_n \mid n \geq 1\}$. We will apply the compactness theorem to Γ.

To do so, we have to show that any finite subset Δ of Γ is satisfiable. Let Δ be an arbitrary finite subset of Γ, and let k be the index of the "largest" formula ϕ_n in Δ. Since there is a model of ψ with at least k elements, $\{\psi, \phi_k\}$ is satisfiable.

But since $\phi_k \rightarrow \phi_n$ is valid for any $n \leq k$, Δ must be satisfiable as well. Now we can invoke compactness and say that Γ is satisfiable by some model $\mathcal{M}$. But if $\mathcal{M}$ is finite, say of size t, then it cannot satisfy ϕ_{t+1}. Thus $\mathcal{M}$ is infinite. $\square$

Intuitively, this theorem says that the concept of "finiteness" is not expressible in predicate logic. Next, we will use compactness to prove that reachability is not expressible in predicate logic.

Theorem (2.26): There is no formula ϕ in predicate logic with free variables u, v and the following property: ϕ is satisfied by a model defined by a directed graph G if and only if there is a path in G from the node associated with u to the node associated with v.

Proof: Suppose that there was such a ϕ. We will derive a contradiction by using ϕ to construct an infinite set Δ which is not satisfiable, even though every finite subset of it is.

Recall that E is the edge relation for G. The formulas in Δ will use E as well as two constants c, c'. We define ϕ_0 as $c = c'$ and

$$\phi_n = \exists x_1 \exists x_2 \ldots \exists x_n (E(c, x_1) \wedge E(x_1, x_2) \wedge \ldots \wedge E(x_{n-1}, c'))$$

Intuitively, the formula ϕ_n says "There is a path from c to c' of length n", so the negation $\neg\phi_n$ says "There is no path from c to c' of length n".

If we substitute c, c' for the free variables u, v in ϕ to obtain $\phi[c/u][c'/v]$, this intuitively says "There is some finite path from c to c'." If ϕ really expresses reachability, then it is true in a model if and only if one of the formulas ϕ_n is true in that model.

Let $\Delta = \{\neg\phi_n \mid n \geq 0\} \cup \{\phi[c/u][c'/v]\}$. By construction, Δ is unsatisfiable, because the first set says "There is no path of any finite length from c to c'" and the second set says that there is one.

But any finite subset of Δ is satisfiable. Such a subset can contain at most a finite subset of $\{\neg\phi_n \mid n \geq 0\}$. Suppose k is the largest index such that $\neg\phi_k$ is in the subset.

Consider the model consisting of a graph which is a single path $v_1, v_2, \ldots, v_{k+1}$, where c is interpreted as v_1 and c' is interpreted as v_{k+1}. Then there aren't paths short enough from v_1 to v_{k+1} to make the ϕ_i false in this model, but there is a path (of length $k+1$) showing that $\phi[c/u][c'/v]$ is true. So this model satisfies the finite subset of Δ.

This is a contradiction to the Compactness Theorem, and therefore the formula ϕ expressing reachability cannot exist. $\square$

Since predicate logic is inadequate to express this important concept, we must go beyond it.

Existential second-order logic

The language of predicate logic we have defined is called **first-order**. Second-order logic extends first-order logic by permitting quantification not just over variables, but over predicate symbols as well. Here is non-reachability expressed in second-order logic:

$$\begin{aligned} \exists P \forall x \forall y \forall z \quad & (P(x,x) \\ \wedge \quad & (P(x,y) \wedge P(y,z) \rightarrow P(x,z)) \\ \wedge \quad & (E(x,y) \rightarrow P(x,y)) \\ \wedge \quad & (\neg P(u,v))) \; . \end{aligned}$$

$$\begin{aligned}\exists P \forall x \forall y \forall z \quad & (P(x,x) \\ \wedge \quad & (P(x,y) \wedge P(y,z) \rightarrow P(x,z)) \\ \wedge \quad & (E(x,y) \rightarrow P(x,y)) \\ \wedge \quad & (\neg P(u,v))) \; .\end{aligned}$$

... how do we read this?

The forumula asserts the existence of a (binary) relation P. The first two clauses tell us that P is reflexive and transitive.

The third clause tells us that P contains the edge relation E; hence P is a reflexive and transitive extension of E.

Thus, P contains the reflexive, transitive closure of E, which is reachability.

The fourth clause asserts that P does not contain (u, v). Hence (u, v) is not in the reflexive, transitive closure of E (otherwise (u, v) would be in P), so v is not reachable from u.

In total, we can read the formula as, “There is a reflexive, transitive extension of the edge relation that does not contain (u, v),” which is equivalent to non-reachability.

By negating this formula, we obtain a formula expressing reachability.

Second-order logic allows arbitrary quantification over predicates. However, we only used an existential quantifier in our definition of non-reachability, and so we have a formula of **existential** second-order logic.

We showed how to express non-reachability in existential second-order logic; the book mentions that reachability is also expressible in existential second-order logic. This is a consequence of a more general, and quite surprising, result.

Earlier, we discussed the fact that the satisfiability problem for propositional logic was “NP-complete” and therefore thought to be hard computationally. In CS 341, you learn that this really means “complete for the class NP”. Intuitively, NP is the class of problems whose solutions are efficiently verifiable. Many problems have this property: it may be hard to find a solution, but if someone gives you one, it is easy to verify that it is a solution.

Fagin proved in 1974 that the set of graph properties in NP are exactly those which can be expressed in existential second-order logic. This is surprising because it relates a notion of efficient computation to one of description with no mention of a model of computation. The field of descriptive complexity explores similar results and their implications (e.g. in databases).

Other proof systems

The other proof systems we briefly discussed for propositional logic (semantic tableaux, sequent calculus, and transformational proofs) can all be extended to propositional logic in a fairly straightforward fashion.

Of these, the most important in practical terms is probably transformational proofs. Recall that the idea, when applied to propositional logic, was to use equivalences in an algebraic fashion. The key advantage was that we could make a substitution of one equivalent subformula for another, whereas for natural deductions, the syntactic changes only happen at the “top level”.

We extend the notion of transformational proof to predicate logic by allowing substitutions based on the quantifier equivalences discussed in this module (and summarized in Theorem 2.13 in the text).

As before, mathematical proofs involving quantification are in practice a mixture of natural deduction and transformational proofs. Notions such as proof by contradiction remain important, and new notions of natural deduction for predicate logic such as $\forall$-introduction become important.

Being able to understand and derive such mathematical proofs requires familiarity with quantifier equivalences as well as knowledge of which possible equivalences fail to hold and what parts of them might be salvageable.

For example, we know that $\forall x(P(x) \vee Q(x)) \not\models \forall x P(x) \vee \forall x Q(x)$. On the other hand, we can show that $\forall x P(x) \vee \forall x Q(x) \models \forall x(P(x) \vee Q(x))$. So a transformation in one direction is possible.

However, it is not always clear in **which** direction the transformation holds. As you proved at the end of Assignment 2, it is possible to have $\phi_1 \models \phi_2$ (rather than full equivalence), and yet have a formula ψ in which the transformation of ϕ_1 to ϕ_2 is **not** valid. It is possible (though we do not discuss it) to characterize the circumstances under which transformation via a one-direction entailment is possible, and the direction in which the transformation must be made, but we will not pursue it here.

Among the most important quantifier equivalences are the “de Morgan”-style ones, such as $\neg\forall x\phi \equiv \exists x\neg\phi$.

Note, however, that this particular equivalence is only a full equivalence under classical reasoning. Under intutionist reasoning we only have the sequent $\exists x\neg\phi \vdash \neg\forall x\phi$. Hence, as intuitionists, we may only reason with the unidirectional entailment $\exists x\neg\phi \models \neg\forall x\phi$ and so we must be careful that transformation of the left-hand side to the right-hand side is valid in the context in which we are performing it.

To illustrate an extreme example of the use of such equivalences, we will quote a central theorem from CS 360 and CS 365, to examine its form (rather than its meaning or proof).

The pumping lemma describes a property of regular languages, sets of strings that are accepted by finite state machines (or equivalently, described by regular expressions). These are first studied in CS 241. It is of the form "If L is regular, then ϕ holds". ϕ happens to be describable in first-order logic using quantifiers.

Here is the form of ϕ, as typically stated in CS 360. Don't worry about the precise meaning.

$$L \in \mathcal{R} \rightarrow \exists n$$
$$\forall s \text{ such that } |s| = n$$
$$\exists x \exists y \exists z \text{ such that } s = xyz$$
$$\forall i \geq 0 \; xy^i z \in L$$

The theorem states a property of regular languages, but it is almost never applied to regular languages. Instead, it is applied to languages believed nonregular, in the contrapositive. Instead of $\psi \rightarrow \phi$, it is used in the form $\neg\phi \rightarrow \neg\psi$. The contrapositive of the pumping lemma looks like this:

$$\neg(\exists n$$
$$\forall s \text{ such that } |s| = n$$
$$\exists x \exists y \exists z \text{ such that } s = xyz$$
$$\forall i \geq 0 \; xy^i z \in L) \rightarrow L \notin \mathcal{R}$$

In order to work with this form, the negation has to be pushed all the way through the nested quantifiers, using the deMorgan-style transformations.

This yields the following formula:

$$
\begin{aligned}
&(\forall n \\
&\qquad \exists s \text{ such that } |s| = n \\
&\qquad\qquad \forall x \forall y \forall z \text{ such that } s = xyz \\
&\qquad\qquad\qquad \exists i \geq 0 \; xy^i z \notin L) \rightarrow L \notin \mathcal{R}.
\end{aligned}
$$

This is the form in which CS 360 students actually work with the pumping lemma in order to prove that a language L is not regular. Without exposure to formal logic, it may not be clear why this form is equivalent to the original statement of the pumping lemma.

What’s next?

The formulas on the previous slides did not conform to our formal definition of formulas in predicate logic, because they made intuitive use of symbols from set theory and notation for strings that cannot be interpreted arbitrarily. Notation from arithmetic and higher mathematics is also important in the proofs we encounter and the specifications we wish to write.

In the next module, we will discuss how these familiar informal notions can be made formal, and cover some rules of thumb about how to identify formal elements in informal descriptions and how to work with them.

Goals of this module

You should understand the semantics of predicate logic, how to apply it for individual formulas, and how to reason about it in general.

You should be able to come up with counterexamples for invalid formulas.

You should understand the meaning of soundness and completeness for predicate logic, and the implications of it as discussed, including the compactness theorem.

You should understand why reachability is not expressible in first-order logic, and how it can be expressed in second-order logic.

You should understand the ideas of transformational proof for predicate logic, and its use in practice.