Semantics, on the other hand, works in the opposite way. To show that ψ is *not* a consequence of Γ is the 'easy' bit: find a model in which all ϕ_i are true, but ψ isn't. Showing that ψ is a consequence of Γ, on the other hand, is harder in principle. For propositional logic, you need to show that every valuation (an assignment of truth values to all atoms involved) that makes all ϕ_i true also makes ψ true. If there is a small number of valuations, this is not so bad. However, when we look at predicate logic, we will find that there are infinitely many valuations, called *models* from hereon, to consider. Thus, in semantics we have a 'negative' characterisation of the logic. We find establishing assertions of the form '$\Gamma \not\vDash \psi$' (ψ is not a semantic entailment of all formulas in Γ) easier than establishing '$\Gamma \vDash \psi$' (ψ is a semantic entailment of Γ), for in the former case we need only talk about one model, whereas in the latter we potentially have to talk about infinitely many.

All this goes to show that it is important to study *both* proof theory *and* semantics. For example, if you are trying to show that ψ is not a consequence of Γ and you have a hard time doing that, you might want to change your strategy for a while by trying to prove the validity of $\Gamma \vdash \psi$. If you find a proof, you know for sure that ψ is a consequence of Γ. If you can't find a proof, then your attempts at proving it often provide insights which lead you to the construction of a counter example. The fact that proof theory and semantics for predicate logic are equivalent is amazing, but it does not stop them having separate roles in logic, each meriting close study.

2.4.1 Models

Recall how we evaluated formulas in propositional logic. For example, the formula $(p \vee \neg q) \to (q \to p)$ is evaluated by computing a truth value (T or F) for it, based on a given valuation (assumed truth values for p and q). This activity is essentially the construction of one line in the truth table of $(p \vee \neg q) \to (q \to p)$. How can we evaluate formulas in predicate logic, e.g.

$$\forall x\, \exists y\, ((P(x) \vee \neg Q(y)) \to (Q(x) \to P(y)))$$

which 'enriches' the formula of propositional logic above? Could we simply assume truth values for $P(x)$, $Q(y)$, $Q(x)$ and $P(y)$ and compute a truth value as before? Not quite, since we have to reflect the meaning of the quantifiers $\forall x$ and $\exists y$, their *dependences* and the actual parameters of P and Q – a formula $\forall x\, \exists y\, R(x, y)$ generally means something else other than $\exists y\, \forall x\, R(x, y)$; why? The problem is that variables are place holders for any, or some, unspecified concrete values. Such values can be of almost any kind: students, birds, numbers, data structures, programs and so on.

Thus, if we encounter a formula $\exists y\,\psi$, we try to find some instance of y (some concrete value) such that ψ holds for that particular instance of y. If this succeeds (i.e. there is such a value of y for which ψ holds), then $\exists y\,\psi$ evaluates to `T`; otherwise (i.e. there is *no* concrete value of y which realises ψ) it returns `F`. Dually, evaluating $\forall x\,\psi$ amounts to showing that ψ evaluates to `T` for *all* possible values of x; if this is successful, we know that $\forall x\,\psi$ evaluates to `T`; otherwise (i.e. there is *some* value of x such that ψ computes `F`) it returns `F`. Of course, such evaluations of formulas require a fixed universe of concrete values, the things we are, so to speak, talking about. Thus, the truth value of a formula in predicate logic depends on, and varies with, the actual choice of values and the meaning of the predicate and function symbols involved.

If variables can take on only finitely many values, we can write a program that evaluates formulas in a compositional way. If the root node of ϕ is $\wedge$, $\vee$, $\rightarrow$ or $\neg$, we can compute the truth value of ϕ by using the truth table of the respective logical connective and by computing the truth values of the subtree(s) of that root, as discussed in Chapter 1. If the root is a quantifier, we have sketched above how to proceed. This leaves us with the case of the root node being a predicate symbol P (in propositional logic this was an atom and we were done already). Such a predicate requires n arguments which have to be terms $t_1, t_2, \dots, t_n$. Therefore, we need to be able to assign truth values to formulas of the form $P(t_1, t_2, \dots, t_n)$.

For formulas $P(t_1, t_2, \dots, t_n)$, there is more going on than in the case of propositional logic. For $n = 2$, the predicate P could stand for something like 'the number computed by t_1 is less than, or equal to, the number computed by t_2.' Therefore, we cannot just assign truth values to P directly without knowing the meaning of terms. We require a *model* of all function and predicate symbols involved. For example, terms could denote *real numbers* and P could denote the relation 'less than or equal to' on the set of real numbers.

Definition 2.14 Let $\mathcal{F}$ be a set of function symbols and $\mathcal{P}$ a set of predicate symbols, each symbol with a fixed number of required arguments. A model $\mathcal{M}$ of the pair $(\mathcal{F}, \mathcal{P})$ consists of the following set of data:

1. A non-empty set A, the universe of concrete values;
2. for each nullary function symbol $f \in \mathcal{F}$, a concrete element $f^{\mathcal{M}}$ of A
3. for each $f \in \mathcal{F}$ with arity $n > 0$, a concrete function $f^{\mathcal{M}} \colon A^n \to A$ from A^n, the set of n-tuples over A, to A; and
4. for each $P \in \mathcal{P}$ with arity $n > 0$, a subset $P^{\mathcal{M}} \subseteq A^n$ of n-tuples over A.

The distinction between f and $f^{\mathcal{M}}$ and between P and $P^{\mathcal{M}}$ is most important. The symbols f and P are just that: symbols, whereas $f^{\mathcal{M}}$ and $P^{\mathcal{M}}$ denote a concrete function (or element) and relation in a model $\mathcal{M}$, respectively.

Example 2.15 Let $\mathcal{F} \stackrel{\text{def}}{=} \{i\}$ and $\mathcal{P} \stackrel{\text{def}}{=} \{R, F\}$; where i is a constant, F a predicate symbol with one argument and R a predicate symbol with two arguments. A model $\mathcal{M}$ contains a set of concrete elements A – which may be a set of states of a computer program. The interpretations $i^{\mathcal{M}}$, $R^{\mathcal{M}}$, and $F^{\mathcal{M}}$ may then be a designated initial state, a state transition relation, and a set of final (accepting) states, respectively. For example, let $A \stackrel{\text{def}}{=} \{a, b, c\}$, $i^{\mathcal{M}} \stackrel{\text{def}}{=} a$, $R^{\mathcal{M}} \stackrel{\text{def}}{=} \{(a, a), (a, b), (a, c), (b, c), (c, c)\}$, and $F^{\mathcal{M}} \stackrel{\text{def}}{=} \{b, c\}$. We informally check some formulas of predicate logic for this model:

1. The formula
$$\exists y\, R(i, y)$$
says that there is a transition from the initial state to some state; this is true in our model, as there are transitions from the initial state a to a, b, and c.
2. The formula
$$\neg F(i)$$
states that the initial state is not a final, accepting state. This is true in our model as b and c are the only final states and a is the intitial one.
3. The formula
$$\forall x \forall y \forall z\, (R(x, y) \wedge R(x, z) \rightarrow y = z)$$
makes use of the equality predicate and states that the transition relation is deterministic: all transitions from any state can go to at most one state (there may be no transitions from a state as well). This is false in our model since state a has transitions to b and c.
4. The formula
$$\forall x \exists y\, R(x, y)$$
states that the model is free of states that deadlock: all states have a transition to some state. This is true in our model: a can move to a, b or c; and b and c can move to c.

Example 2.16 Let $\mathcal{F} \stackrel{\text{def}}{=} \{e, \cdot\}$ and $\mathcal{P} \stackrel{\text{def}}{=} \{\leq\}$, where e is a constant, $\cdot$ is a function of two arguments and $\leq$ is a predicate in need of two arguments as well. Again, we write $\cdot$ and $\leq$ in infix notation as in $(t_1 \cdot t_2) \leq (t \cdot t)$.

The model $\mathcal{M}$ we have in mind has as set A all binary strings, finite words over the alphabet $\{0, 1\}$, including the empty string denoted by ϵ. The interpretation $e^{\mathcal{M}}$ of e is just the empty word ϵ. The interpretation $\cdot^{\mathcal{M}}$ of $\cdot$ is the concatenation of words. For example, 0110 $\cdot^{\mathcal{M}}$ 1110 equals 01101110. In general, if $a_1a_2 \dots a_k$ and $b_1b_2 \dots b_n$ are such words with $a_i, b_j \in \{0, 1\}$, then $a_1a_2 \dots a_k \cdot^{\mathcal{M}} b_1b_2 \dots b_n$ equals $a_1a_2 \dots a_kb_1b_2 \dots b_n$. Finally, we interpret $\leq$ as the prefix ordering of words. We say that s_1 is a prefix of s_2 if there is a binary word s_3 such that $s_1 \cdot^{\mathcal{M}} s_3$ equals s_2. For example, 011 is a prefix of 011001 and of 011, but 010 is neither. Thus, $\leq^{\mathcal{M}}$ is the set $\{(s_1, s_2) \mid s_1 \text{ is a prefix of } s_2\}$. Here are again some informal model checks:

1. In our model, the formula
$$\forall x\, ((x \leq x \cdot e) \wedge (x \cdot e \leq x))$$
says that every word is a prefix of itself concatenated with the empty word and conversely. Clearly, this holds in our model, for $s \cdot^{\mathcal{M}} \epsilon$ is just s and every word is a prefix of itself.
2. In our model, the formula
$$\exists y\, \forall x\, (y \leq x)$$
says that there exists a word s that is a prefix of every other word. This is true, for we may chose ϵ as such a word (there is no other choice in this case).
3. In our model, the formula
$$\forall x\, \exists y\, (y \leq x)$$
says that every word has a prefix. This is clearly the case and there are in general multiple choices for y, which are dependent on x.
4. In our model, the formula $\forall x\, \forall y\, \forall z\, ((x \leq y) \to (x \cdot z \leq y \cdot z))$ says that whenever a word s_1 is a prefix of s_2, then s_1s has to be a prefix of s_2s for every word s. This is clearly not the case. For example, take s_1 as 01, s_2 as 011 and s to be 0.
5. In our model, the formula
$$\neg \exists x\, \forall y\, ((x \leq y) \to (y \leq x))$$
says that there is no word s such that whenever s is a prefix of some other word s_1, it is the case that s_1 is a prefix of s as well. This is true since there cannot be such an s. Assume, for the sake of argument, that there were such a word s. Then s is clearly a prefix of $s0$, but $s0$ cannot be a prefix of s since $s0$ contains one more bit than s.

It is crucial to realise that the notion of a model is extremely liberal and open-ended. All it takes is to choose a non-empty set A, whose elements model real-world objects, and a set of concrete functions and relations, one for each function, respectively predicate, symbol. The only mild requirement imposed on all of this is that the concrete functions and relations on A have the same number of arguments as their syntactic counterparts.

However, you, as a designer or implementor of such a model, have the responsibility of choosing your model wisely. Your model should be a sufficiently accurate picture of whatever it is you want to model, but at the same time it should abstract away (= ignore) aspects of the world which are irrelevant from the perspective of your task at hand.

For example, if you build a database of family relationships, then it would be foolish to interpret *father-of*(x, y) by something like 'x is the daughter of y.' By the same token, you probably would not want to have a predicate for 'is taller than,' since your focus in this model is merely on relationships defined by birth. Of course, there are circumstances in which you may want to add additional features to your database.

Given a model $\mathcal{M}$ for a pair $(\mathcal{F}, \mathcal{P})$ of function and predicate symbols, we are now almost in a position to formally compute a truth value for all formulas in predicate logic which involve only function and predicate symbols from $(\mathcal{F}, \mathcal{P})$. There is still one thing, though, that we need to discuss. Given a formula $\forall x\, \phi$ or $\exists x\, \phi$, we intend to check whether ϕ holds for all, respectively some, value a in our model. While this is intuitive, we have no way of expressing this in our syntax: the formula ϕ usually has x as a free variable; $\phi[a/x]$ is well-intended, but ill-formed since $\phi[a/x]$ is *not* a logical formula, for a is not a term but an element of our model.

Therefore we are forced to interpret formulas *relative to an environment*. You may think of environments in a variety of ways. Essentially, they are look-up tables for all variables; such a table l associates with every variable x a value $l(x)$ of the model. So you can also say that environments are functions $l \colon \mathsf{var} \to A$ from the set of variables var to the universe of values A of the underlying model. Given such a look-up table, we can assign truth values to all formulas. However, for some of these computations we need *updated* look-up tables.

Definition 2.17 A look-up table or environment for a universe A of concrete values is a function $l \colon \mathsf{var} \to A$ from the set of variables var to A. For such an l, we denote by $l[x \mapsto a]$ the look-up table which maps x to a and any other variable y to $l(y)$.

Finally, we are able to give a semantics to formulas of predicate logic. For propositional logic, we did this by computing a truth value. Clearly, it suffices to know in which cases this value is $\mathtt{T}$.

Definition 2.18 Given a model $\mathcal{M}$ for a pair $(\mathcal{F}, \mathcal{P})$ and given an environment l, we define the satisfaction relation $\mathcal{M} \vDash_l \phi$ for each logical formula ϕ over the pair $(\mathcal{F}, \mathcal{P})$ and look-up table l by structural induction on ϕ. If $\mathcal{M} \vDash_l \phi$ holds, we say that ϕ computes to $\mathtt{T}$ in the model $\mathcal{M}$ with respect to the environment l.

P: If ϕ is of the form $P(t_1, t_2, \dots, t_n)$, then we interpret the terms $t_1, t_2, \dots, t_n$ in our set A by replacing all variables with their values according to l. In this way we compute concrete values $a_1, a_2, \dots, a_n$ of A for each of these terms, where we interpret any function symbol $f \in \mathcal{F}$ by $f^{\mathcal{M}}$. Now $\mathcal{M} \vDash_l P(t_1, t_2, \dots, t_n)$ holds iff $(a_1, a_2, \dots, a_n)$ is in the set $P^{\mathcal{M}}$.
$\forall x$: The relation $\mathcal{M} \vDash_l \forall x\, \psi$ holds iff $\mathcal{M} \vDash_{l[x \mapsto a]} \psi$ holds for all $a \in A$.
$\exists x$: Dually, $\mathcal{M} \vDash_l \exists x\, \psi$ holds iff $\mathcal{M} \vDash_{l[x \mapsto a]} \psi$ holds for some $a \in A$.
$\neg$: The relation $\mathcal{M} \vDash_l \neg\psi$ holds iff it is not the case that $\mathcal{M} \vDash_l \psi$ holds.
$\vee$: The relation $\mathcal{M} \vDash_l \psi_1 \vee \psi_2$ holds iff $\mathcal{M} \vDash_l \psi_1$ or $\mathcal{M} \vDash_l \psi_2$ holds.
$\wedge$: The relation $\mathcal{M} \vDash_l \psi_1 \wedge \psi_2$ holds iff $\mathcal{M} \vDash_l \psi_1$ and $\mathcal{M} \vDash_l \psi_2$ hold.
$\rightarrow$: The relation $\mathcal{M} \vDash_l \psi_1 \rightarrow \psi_2$ holds iff $\mathcal{M} \vDash_l \psi_2$ holds whenever $\mathcal{M} \vDash_l \psi_1$ holds.

We sometimes write $\mathcal{M} \nvDash_l \phi$ to denote that $\mathcal{M} \vDash_l \phi$ does not hold.

There is a straightforward inductive argument on the height of the parse tree of a formula which says that $\mathcal{M} \vDash_l \phi$ holds iff $\mathcal{M} \vDash_{l'} \phi$ holds, whenever l and l' are two environments which are identical on the set of free variables of ϕ. In particular, if ϕ has *no* free variables at all, we then call ϕ a *sentence*; we conclude that $\mathcal{M} \vDash_l \phi$ holds, or does not hold, regardless of the choice of l. Thus, for sentences ϕ we often elide l and write $\mathcal{M} \vDash \phi$ since the choice of an environment l is then irrelevant.

Example 2.19 Let us illustrate the definitions above by means of another simple example. Let $\mathcal{F} \overset{\text{def}}{=} \{\mathsf{alma}\}$ and $\mathcal{P} \overset{\text{def}}{=} \{\mathsf{loves}\}$ where alma is a constant and loves a predicate with two arguments. The model $\mathcal{M}$ we choose here consists of the privacy-respecting set $A \overset{\text{def}}{=} \{a, b, c\}$, the constant function $\mathsf{alma}^{\mathcal{M}} \overset{\text{def}}{=} a$ and the predicate $\mathsf{loves}^{\mathcal{M}} \overset{\text{def}}{=} \{(a,a), (b,a), (c,a)\}$, which has two arguments as required. We want to check whether the model $\mathcal{M}$ satisfies

None of Alma's lovers' lovers love her.

First, we need to express the, morally worrying, sentence in predicate logic. Here is such an encoding (as we already discussed, different but logically equivalent encodings are possible):

$$\forall x\, \forall y\, (\mathsf{loves}(x, \mathsf{alma}) \wedge \mathsf{loves}(y, x) \rightarrow \neg\mathsf{loves}(y, \mathsf{alma}))\ . \tag{2.8}$$

Does the model $\mathcal{M}$ satisfy this formula? Well, it does not; for we may choose a for x and b for y. Since (a, a) is in the set $\mathsf{loves}^{\mathcal{M}}$ and (b, a) is in the set $\mathsf{loves}^{\mathcal{M}}$, we would need that the latter does not hold since it is the interpretation of $\mathsf{loves}(y, \mathsf{alma})$; this cannot be.

And what changes if we modify $\mathcal{M}$ to $\mathcal{M}'$, where we keep A and $\mathsf{alma}^{\mathcal{M}}$, but redefine the interpretation of loves as $\mathsf{loves}^{\mathcal{M}'} \stackrel{\text{def}}{=} \{(b, a), (c, b)\}$? Well, now there is exactly one lover of Alma's lovers, namely c; but c is not one of Alma's lovers. Thus, the formula in (2.8) holds in the model $\mathcal{M}'$.

2.4.2 Semantic entailment

In propositional logic, the semantic entailment $\phi_1, \phi_2, \dots, \phi_n \vDash \psi$ holds iff: whenever all $\phi_1, \phi_2, \dots, \phi_n$ evaluate to $\mathtt{T}$, the formula ψ evaluates to $\mathtt{T}$ as well. How can we define such a notion for formulas in predicate logic, considering that $\mathcal{M} \vDash_l \phi$ is indexed with an environment?

Definition 2.20 Let Γ be a (possibly infinite) set of formulas in predicate logic and ψ a formula of predicate logic.

1. Semantic entailment $\Gamma \vDash \psi$ holds iff for all models $\mathcal{M}$ and look-up tables l, whenever $\mathcal{M} \vDash_l \phi$ holds for all $\phi \in \Gamma$, then $\mathcal{M} \vDash_l \psi$ holds as well.
2. Formula ψ is satisfiable iff there is some model $\mathcal{M}$ and some environment l such that $\mathcal{M} \vDash_l \psi$ holds.
3. Formula ψ is valid iff $\mathcal{M} \vDash_l \psi$ holds for all models $\mathcal{M}$ and environments l in which we can check ψ.
4. The set Γ is consistent or satisfiable iff there is a model $\mathcal{M}$ and a look-up table l such that $\mathcal{M} \vDash_l \phi$ holds for all $\phi \in \Gamma$.

In predicate logic, the symbol $\vDash$ is overloaded: it denotes model checks '$\mathcal{M} \vDash \phi$' and semantic entailment '$\phi_1, \phi_2, \dots, \phi_n \vDash \psi$.' Computationally, each of these notions means trouble. First, establishing $\mathcal{M} \vDash \phi$ will cause problems, if done on a machine, as soon as the universe of values A of $\mathcal{M}$ is infinite. In that case, checking the sentence $\forall x\, \psi$, where x is free in ψ, amounts to verifying $\mathcal{M} \vDash_{[x \mapsto a]} \psi$ for infinitely many elements a.

Second, and much more seriously, in trying to verify that $\phi_1, \phi_2, \dots, \phi_n \vDash \psi$ holds, we have to check things out for *all possible models*, all models which are equipped with the right structure (i.e. they have functions and predicates with the matching number of arguments). This task is impossible to perform mechanically. This should be contrasted to the situation in propositional logic, where the computation of the truth tables for the propositions involved was the basis for computing this relationship successfully.