The main problem we face now, when we try to extend the semantics of a truth table to predicate logic, is how to consistently manage variables. If we have just one predicate in a formula, such as $\exists x P(x)$, we can imagine a process for assigning each value in the universe to $x$ until we find one that has the property $P$. Trying to do this for a more complicated formula turns out to require some logical machinery.

This difficulty in handling variables can be resolved if we turn from intensional definitions of predicate and instead use extensional definitions. This changes our point of view and, for a computer, means that determining whether a predicate holds will be the process of determining whether a given thing is in a set.

We saw this in the example of a model for a finite-state machine. The relation $R$ was defined extensionally as a set, so that determining whether the relation held between two elements of the universe $A$ was a matter of determining whether a given pair was in the set $R$.

This is the insight of extensional definition: a predicate is a set. For an intensionally defined predicate $P^M$ in a model $M$, we can give the extensional definition

$$P^M \overset{\text{df}}{=} \{ x | P^M(x) \} \quad (26.1)$$

We can extensionally define a predicate by a set rule, or by enumeration, according to the application of interest.

For example, there are many equivalent definitions of the set of odd integers. Intensionally, we could say that an odd integer is an integer that passes a test of not being divisible by 2. This might be something like

$$P^M(x) : x \equiv 1 \mod 2$$

Extensionally, we could say that the set of odd integers is the set of numbers that are the results of adding 1 to an even number, which could be

$$P^M = \{ x | x = (2k + 1) \text{ for some } k \}$$

or the set of all integers minus the set of all even integers, which could be

$$P^M = \mathbb{Z} \setminus 2\mathbb{N}$$

From now on, we will assume that any predicate or relation is a set.
To be able to properly describe predicates, we must be able to handle variables. For example, consider the implication

\[ P(c) \rightarrow \exists x \ P(x) \]

If we have a model \( M \), and it has the constant \( c \in A \), we can write the antecedent as \( P^M(c) \).

When we try to model the consequent, we observe that the variable \( x \) is free in the formula \( P(x) \). We can model the predicate, because it works out to a set by an extensional definition, but the symbolic string \( P^M(x^M) \) does not make sense. The key question this raises is, “What is a model for a variable?”

To be able to properly handle variables, we introduce the concept of an environment. We already have this concept from computing, where the environment is the context within which a computation happens; it is the hardware plus the software. In predicate logic, the environment is the set of variables and how to map the variables to the universe of discourse \( A \). We will formalize an environment as a mapping, which has two parts.

We will use the symbol \( l \) to represent an environment. The domain of the mapping \( l \) is the set of valid inputs to the map, which is an extensional definition. In mathematics this set would be written as “\( \text{dom } l \)” but in our textbook the set is written as \( \text{var} \).

The set \( \text{var} \) is the name of all variables of a set of formulas. We can see, immediately, that the environment and the formulas are closely related; this makes sense, because in computing when we declare a new global variable we are adding it to the environment. For now, let us suppose that an environment can model all of the variables in our formulas of interest.

An environment is an extensionally defined mapping. By this, we mean that an environment \( l \) is a set of tuples. The first element of a tuple is a variable name and the second element of a tuple is a value from the universe of discourse \( A \). For example, if we have two variables \( y \) and \( z \), and the universe is three values \( A = \{a, b, c\} \), then we could have as an environment

\[
\begin{align*}
\text{var} & \overset{\text{def}}{=} \{y, z\} \\
l & \overset{\text{def}}{=} \{(y, c), (z, a)\}
\end{align*}
\] (26.2)

The environment \( l \) is a mapping, so it can be invoked as a function. When we evaluate the \( l \) of Environment 26.2 on variable \( y \), we get the value \( c \), or

\[ l(y) = c \]

It is easy to extend an environment in predicate logic. For example, in Environment 26.2, we do not have a variable \( x \). If we needed this new variable to be in the environment, and we want it to map to the value \( c \), we can add this information into the environment by the statement

\[ l[x \mapsto c] \] (26.3)
Combining Environment 26.2 with Extension 26.3, we get a new environment

\[ \text{var} \overset{\text{df}}{=} \{y, z, x\} \]
\[ l' \overset{\text{df}}{=} \{(y, c), (z, a), (x, c)\} \] (26.4)

We can see that the mappings \( l \) and \( l' \) each have a finite set as the domain and are “into” mappings, so the range is also finite. One useful way to represent a finite mapping—especially one that is defined extensionally as a set—is as a look-up table. We can now see that the examples of Environment 26.2 and Environment 26.4 are exactly look-up tables: given a variable in the domain of the mapping, we can look up the element from the universe of discourse \( A \) that is the model for the variable.

This now allows us to define an environment and how to extend an environment to model a new variable.

**Definition:** Environment of Predicate Semantics

An environment for a universe \( A \) is a mapping \( l : \text{var} \rightarrow A \) from the set of variables \( \text{var} \) to values in \( A \).

An extended environment of an environment \( l \) is a mapping \( l[x \mapsto a] \) that maps a new variable \( x \) to \( a \in A \) and that maps any distinct variable \( y \) to \( l(y) \).

To see how to use an environment, consider the partial model of base-4 arithmetic that we introduced in a previous class. We had \( A = \{0, 1, 2, 3\} \) and a predicate \( \overline{P}^M \) that was interpreted as the set \( P^M \) of even numbers in \( A \).

Is there an environment in which \( \overline{P}^M(x) \) holds? That is, is there a mapping of \( x \) into \( P^M \)?

Such an environment can be created from an existing environment, which could be the empty environment. We can use either of the semantic statements

\[ l[x \mapsto 0] \]
\[ l[x \mapsto 2] \]

to accomplish our goal. In the base-4 example, we also had a function \( f^M(\cdot) \) that added 1 to its input, modulo 4; the mapping \( l(x) \) gives us the ability to reason about \( P^M(f^M(x)) \) and other, more elaborate, formulas.

What we have done here is that we have found a model and an environment that a formula \( \phi \) satisfies. We now can provide a definition of satisfaction of a formula in predicate logic.

We will begin with an abbreviation that is not in the textbook but that is commonly used.

**Definition:** Interpretation

An interpretation \( \mathcal{I} \) is a model \( \mathcal{M} \) combined with an environment \( l \).
The concept of formula satisfaction is defined recursively. In English, the base case is: a predicate \( P \) is satisfied in a model means that, when we map its terms to the universe \( A \) using the environment \( l \), the resulting values are in the set \( P \). The base case needs to accommodate an \( n \)-ary predicate, and the predicate needs to be extensionally defined as a set.

The notation for the base case is that, for an interpretation \( I \), we represent an \( n \)-ary predicate \( P(t_1, t_2, \ldots, t_n) \) being satisfied for terms \( t_1, t_2, \ldots, t_n \) as

\[
\mathcal{M} \models _l P(t_1, t_2, \ldots, t_n)
\]

From predicates, we build up the concept of satisfaction in two ways. The first is, in parallel to propositional logic, we define satisfaction for the logical operators \( \neg, \land, \lor, \to \). The second, specific to predicate logic, is that we define satisfaction for universal quantification \( \forall x \) and for existential quantification \( \exists x \).

**Definition:** Formula Satisfaction in an Interpretation

Given an interpretation \( \mathcal{I} \), the satisfaction relation \( \mathcal{M} \models _l \phi \) holds for \( \phi \) means:

- If \( \phi \) is \( P(t_1, t_2, \ldots, t_n) \), calculate the value of each term \( t_i \) using the mapping \( l \) to find the value \( t_i = a_i \in A \); in each calculation, use the model of \( f \in \mathcal{F} \) as \( f^M \). Then \( \mathcal{M} \models _l \phi \) holds means that \((a_1, a_2, \ldots, a_n) \in P^M\).

An equivalent statement is: \( \mathcal{M} \models _l \phi \) evaluates to \( T \) if \((a_1, a_2, \ldots, a_n) \in P^M\) and evaluates to \( F \) if \((a_1, a_2, \ldots, a_n) \notin P^M\).

**Otherwise,** \( \mathcal{M} \models _l \phi \) holds means that one of these cases evaluates to \( T \):

<table>
<thead>
<tr>
<th>Case</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{M} \models _l \neg \psi ) evaluates to ( T )</td>
<td>( \mathcal{M} \models _l \psi ) does not evaluate to ( T )</td>
</tr>
<tr>
<td>( \mathcal{M} \models _l \psi_1 \land \psi_2 ) evaluates to ( T )</td>
<td>( \mathcal{M} \models _l \psi_1 ) evaluates to ( T ) and ( \mathcal{M} \models _l \psi_2 ) evaluates to ( T )</td>
</tr>
<tr>
<td>( \mathcal{M} \models _l \psi_1 \lor \psi_2 ) evaluates to ( T )</td>
<td>( \mathcal{M} \models _l \psi_1 ) evaluates to ( T ) or ( \mathcal{M} \models _l \psi_2 ) evaluates to ( T )</td>
</tr>
<tr>
<td>( \mathcal{M} \models _l \psi_1 \to \psi_2 ) evaluates to ( T )</td>
<td>( \mathcal{M} \models _l \psi_2 ) evaluates to ( T ) whenever ( \mathcal{M} \models _l \psi_1 ) evaluates to ( T )</td>
</tr>
<tr>
<td>( \mathcal{M} \models _l \forall x \psi ) evaluates to ( T )</td>
<td>( \mathcal{M} \models _{l[x \mapsto a]} \psi ) evaluates to ( T ) for all ( a \in A )</td>
</tr>
<tr>
<td>( \mathcal{M} \models _l \exists x \psi ) evaluates to ( T )</td>
<td>( \mathcal{M} \models _{l[x \mapsto a]} \psi ) evaluates to ( T ) for some ( a \in A )</td>
</tr>
</tbody>
</table>

An important special situation for satisfaction occurs when a formula \( \phi \) has no free variables. In such a situation, the formula \( \phi \) either holds or does not hold in model \( \mathcal{M} \) regardless of the choice of the environment \( l \). This is because, having no free variables, the assignment of variables in the environment has no effect on the semantics of \( \phi \). To summarize this situation:

- If \( \phi \) has no free variables, it is called a *sentence* in predicate logic;
- A sentence holds or does not hold in a model \( \mathcal{M} \) independent of the choice of the environment \( l \), so we can use the abbreviation \( \mathcal{M} \models \phi \) for sentence \( \phi \).
**Example:** Base-4 arithmetic

Our model of base-4 arithmetic had two predicates because $P = \{P, Q\}$. We will consider three closely related formulas; a theorem, a sentence that is satisfied in the model, and a formula with a free variable.

Consider the English statement

Every number is even or odd

We can translate this in at least two distinct ways. The first is to say that every number is even or is not even. This gives us the sentence

$$\forall x (P(x) \lor \neg P(x)) \quad (26.5)$$

We can easily verify that Formula 26.5 is a theorem, and that it is satisfied in the model $M$.

A second way of translating the English statement is to use both predicates, one for the even numbers and one for the odd numbers. This gives us the sentence

$$\forall x (P(x) \lor Q(x)) \quad (26.6)$$

To determine whether Formula 26.6 holds, we would have to apply the definition of $|=_{i}$ recursively. This recursion is only of depth 2, so it can be expanded in-line. The applications of the definition of satisfaction look like:

1. for each $a \in A$:
2. assign $x$ with value $a$
3. evaluate $\lor$:
   4. evaluate $a \in P$
   5. evaluate $a \in Q$
   6. evaluates to $T$ if Line 4 is $T$ or Line 5 is $T$
   7. evaluates to $T$ if every Line 6 evaluates to $T$

In Line 4 and Line 5, we have used the extensional definitions of the predicates. Line 6 is a reminder to us of the definition of “evaluates” for disjunction, and Line 7 is a reminder of universal quantification. For this model, the sentence in Formula 26.6 is satisfied.

Students should verify that Formula 26.6 does not hold for a model $M'$ in which we use a modified predicate

$$Q' = \{0, 1, 3\}$$

The theorem of Formula 26.5 still holds under this alternate model $M'$. 

109
Next, consider the English statement

The number \( x \) is even or odd

We will use the two-predicate translation, parallel to Formula 26.6, so that we have the formula

\[
P(x) \lor Q(x)
\]  
(26.7)

The \( \phi \) of Formula 26.7 is not a sentence. Examining it, we see that the variable \( x \) is free in \( \phi \). The model \( M \) now needs an environment \( l \) for us to be able to determine whether or not Formula 26.7 holds in \( M \).

Let us consider two environments, \( l_1 \) and \( l_2 \), within this model \( M \) of base-4 arithmetic. Each of these is an assignment of the variable \( x \) to a value in \( A \). We can pick

\[
l_1 \overset{df}{=} \{(x, 0)\}
\]  
(26.8)

\[
l_2 \overset{df}{=} \{(x, 1)\}
\]  
(26.9)

When we apply the definition of satisfaction to Formula 26.7, we get

1. In environment \( l_1 \), assign \( x \) with value \( l_1(x) = 0 \)
2. evaluate \( \lor \):
3. evaluate \( 0 \in P \)
4. evaluate \( 0 \in Q \)
5. evaluates to \( T \) if Line 3 is \( T \) or Line 4 is \( T \)

As we expected from the English statement, Formula 26.7 is satisfied in Model \( M \) with environment \( l_1 \).

Students should verify that, using environment \( l_2 \), Formula 26.7 is also satisfied.

A simple but useful exercise is to find a formula that is satisfied in environment \( l_1 \) and is not satisfied in environment \( l_2 \).