Predicate logic is propositional logic with several new symbol types described below. These additional symbols give predicate logic greater expressive power than propositional logic. Specifically, the new symbols allow predicate logic statements to differentiate among individual objects in some domain of discourse, to refer to properties of objects or tuples of objects, and to quantify the scope of a statement within the domain of discourse. These notes discuss the new ingredients and provide an introduction to first-order predicate logic.

1 New syntactical elements in predicate logic

- **Domain constants and variables** are symbols that refer to objects; for example, if \( x \) is a domain variable, then \( x \) could refer to a number, or a graph, or a program, depending on the type of objects in the domain being described; domain constants are symbols that refer to a specific domain object, such as 0 or Alan Turing.

- **Predicate symbols** (also known as relation symbols) are symbols that refer to properties of individual objects (in the case of unary relations), or to properties of tuples of objects (in the case of \( n \)-ary relations, for \( n \geq 2 \) or higher); for example, if \( P \) is a unary relation symbol, then \( P \) could refer to the property of being yellow, or being an integer, or being employable, whereas a binary relation symbol could refer to a binary relation such as “precedes” or “is a parent of”; in this case, if \( x \) is a domain variable, then \( P(x) \) corresponds to the statement that \( x \) has property \( P \).

- **Function symbols** refer to functions of tuples of objects; for example, a two-parameter function symbol \( f \) could refer to a function such as \( f(x, y) = x^2 + y^2 \) in the case of a numerical object domain, or \( f(a, b) = \) the most recent common ancestor of \( a, b \) in the case of a domain of people.

- **Quantifiers** are symbols that allow describing the extent of the object domain to which a particular logical statement applies. The most commonly used quantifiers, and the only ones in the case of classical first-order predicate logic, are the universal quantifier \( \forall \) (“for all”), and the existential quantifier \( \exists \) (“there exists”); for example, the logical statement \( \forall x \ R(x) \) states that all objects in the domain have property \( R \) (this assumes that \( R \) is a unary predicate symbol), so that, if the domain is the set of integers, for
example, that statement could be interpreted as stating “every integer has a prime divisor”, if \( R(x) \) means that \( x \) has a prime divisor; an alternative description of the existence of prime divisors would be \( \forall x \exists y \text{IsPrime}(y) \land \text{Divides}(y, x) \) (this assumes that \text{IsPrime} and \text{Divides} are relation symbols of the right arity).

Notes.

1. The equality symbol = is usually thrown in “for free”, as a binary relation symbol with the usual meaning.

2. Constants are often not treated as a separate type of syntactical ingredient, but rather as functions with no parameters.

3. The specific constant, relation, and function symbols used vary with the application. These symbols are often referred to as non-logical symbols, to differentiate them from the quantifiers and propositional logic symbols that are always included. Hence, one often speaks not of predicate logic in general, but instead of a particular predicate logic language that includes specific instances of the non-logical symbols.

4. In first-order logic, relation symbols, function symbols, and quantifiers can only be applied to domain variables, not to any other symbols. In particular, it is not possible to quantify over relation symbols. For example, assuming that the domain consists of integers, so that function symbols refer to functions on integers, the statement “every function of a single variable is zero at some point” cannot be expressed as a first-order predicate logic statement. The natural, tempting, approach would be to write:

\[
\forall f (\text{IsFunction}(f) \rightarrow \exists x (f(x) = 0))
\]

However, this uses a relation symbol \text{IsFunction} that applies to function symbols, as well as existential quantification over the function symbol \( f \).

1.1 Examples of predicate logic statements

We give a natural language version of the statement in each case, followed by a predicate logic version in a suitable specific predicate logic language.

- The binary relation \( LT \) is a linear ordering of the elements of the domain.

\[
\forall x \forall y \forall z (\neg LT(x, x) \land (LT(x, y) \lor LT(y, x)) \land (LT(x, y) \land LT(y, z) \rightarrow LT(x, z)))
\]

- There is a largest element.

\[
\exists x \forall y \text{LT}(y, x)
\]

- For every element, there is some element that is larger than it.

\[
\forall y \exists x \text{LT}(y, x)
\]
NEW SYNTACTICAL ELEMENTS IN PREDICATE LOGIC

- The domain is not empty.
  \[ \exists x \ (x = x) \]

- Every green object is shiny.
  \[ \forall x \ (\text{IsGreen}(x) \rightarrow \text{IsShiny}(x)) \]

- Every quadratic polynomial has a root.
  \[ \forall a \forall b \forall c \ (\neg (a = 0) \rightarrow \exists x \ \text{plus}(\times(a, \text{pow}(x, 2)), \times(b, x), c) = 0) \]

If we relax the syntactical rules a bit to allow infix notation (function symbol sandwiched between the operands), we can rewrite the above as:

\[ \forall a \forall b \forall c \ (\neg (a = 0) \rightarrow \exists x \ a \times x \times x + b \times x + c = 0) \]

### 1.2 Terms and formulas

Notice that predicate logic allows for two distinct types of syntactical constructs: a term is a symbolic expression that refers to a domain object, while a formula is a symbolic statement about the domain. Formulas, unlike terms, have a truth value, just like the propositions of propositional logic. These concepts are similar to the concepts of expression and statement in programming languages such as Java or Python.

**Terms.** The terms of predicate logic are defined by recursion:

- (Basis) All constant symbols and domain variables are terms.

- (Recursive step) If \( f \) is an \( n \)-parameter function symbol, and if \( t_1, \ldots, t_n \) are terms, then \( f(t_1, \ldots, t_n) \) is a term.

**Formulas.** The formulas of predicate logic are also defined by recursion:

- (Basis) If \( P \) is an \( n \)-ary relation symbol and if \( t_1, \ldots, t_n \) are terms, then \( P(t_1, \ldots, t_n) \) is a formula. The basis formulas are sometimes called atomic formulas.

- (Recursive step) If \( \phi \) and \( \psi \) are formulas, and if \( x \) is a domain variable, then so is each of the following:
  \[
  (\neg \phi) \\
  (\phi \land \psi) \\
  (\phi \lor \psi) \\
  (\phi \rightarrow \psi) \\
  (\forall x \phi) \\
  (\exists x \phi)
  \]
2 Predicate logic semantics

On their own, predicate logic formulas are disembodied statements about some unspecified world. An interpretation provides the link between the syntax of a formula, on one hand, and a particular domain of interest, on the other.

2.1 Structures

Since predicate logic terms (in the technical sense discussed earlier) refer to domain elements, it is natural that an interpretation is defined by tying these terms to objects in some specific domain. An interpretation of the terms of predicate logic is often referred to as a structure or a model. The semantics of first-order logic is often called model theory.

Definition 2.1. A structure for a first-order logic language is a pair \( \mathfrak{D} = (D, I) \), where \( D \) is a set called the domain, and \( I \) is a collection of interpretations of the domain-related symbols of the language, as follows:

- For each constant symbol \( c \) of the language, there is a specific element \( I(c) = \bar{c} \in D \)
- For each \( n \)-ary predicate symbol \( P \), there is an \( n \)-ary relation \( I(P) = \bar{P} \subseteq D^n \)
- For each \( n \)-ary function symbol \( f \), there is an \( n \)-ary function \( I(f) = \bar{f} : D^n \to D \)

An interpretation \( I(t) = \bar{t} \) can now be found for each term \( t \) of the language, by recursion. Writing this down in detail is left as an important exercise.

Examples

1. Consider the predicate logic language that has as its non-logical symbols a constant symbol \textit{Neutral} and a binary (two-parameter) function symbol \textit{op}. Two possible structures for this language are the following:

   \begin{enumerate}
   \item The domain \( \mathbb{Z} \) of integers, with interpretation \( \text{Neutral} = 0 \), and \( \text{op}(n, m) = n + m \) for all \( n, m \in \mathbb{Z} \).
   \item The domain \([0, 1]\) of real numbers between 0 and 1, with interpretation \( \overline{\text{Neutral}} = 0.5 \), and \( \text{op}(x, y) = \max(x, y) \) for all \( x, y \in [0, 1] \).
   \end{enumerate}

2.2 Truth values with respect to a structure

Given a specific domain and interpretation in a structure \( \mathfrak{D} \), predicate logic formulas represent statements about that structure. Each such statement has an associated truth value with respect to \( \mathfrak{D} \). In classical predicate logic, the allowable truth values are \textit{true} and \textit{false}. If a formula \( \phi \) is \textit{true} in a structure \( \mathfrak{D} \), one says that \( \phi \) holds in \( \mathfrak{D} \) or that \( \mathfrak{D} \) satisfies \( \phi \), and writes \( \mathfrak{D} \models \phi \). Truth values are assigned by recursion, as described below.
Table 1: Tic-tac-toe board.

<table>
<thead>
<tr>
<th></th>
<th>O</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

- (Basis) If $P$ is an $n$-ary relation symbol and if $t_1, \ldots, t_n$ are terms, then the truth value of the atomic formula $P(t_1, \ldots, t_n)$ is true if $P(t_1, \ldots, t_n)$, that is, if the interpretation of $P$ in the domain structure holds for the tuple consisting of the interpretations of $t_1, \ldots, t_n$ in the domain structure. Otherwise, the truth value of the atomic formula $P(t_1, \ldots, t_n)$ is false.\(^1\)

- (Recursive step) If $\phi$ and $\psi$ are formulas, and if $x$ is a domain variable, then truth values are assigned to the formulas below as explained in each case:
  - $(\neg \phi)$ has the opposite truth value of $\phi$
  - $(\phi \land \psi), (\phi \lor \psi), (\phi \rightarrow \psi)$ each has the truth value determined by its truth table
  - $\forall x \phi$ has the truth value true only if for every element $d \in D$ of the domain, the truth value of the formula obtained by substituting $d$ for every occurrence of $x$ in $\phi$ has the truth value true.\(^2\) The truth value of $\forall x \phi$ is false otherwise.
  - $\exists x \phi$ has the truth value true only if for some element $d^* \in D$ of the domain, the truth value of the formula obtained by substituting $d^*$ for every occurrence of $x$ in $\phi$ has the truth value true. The truth value of $\exists x \phi$ is false otherwise.

2.2.1 Example: Tic-Tac-Toe

Consider modeling the board game Tic-Tac-Toe, which is a two-player played on a $3 \times 3$ board as shown in Table 2.2.1. The board is initially empty, and players take turns. On each turn, the player marks a single square of his / her choice. One player uses an $X$, and the other an $O$. The first player to succeed in placing three marks in the same row, column, or diagonal wins the game.

Non-logical symbols. We can model the board by defining 18 predicates $IsX(i, j)$ and $IsO(i, j)$, one pair for each of the nine squares of the board. The numbers $i$ and $j$ represent the row and column in each case. In the interpretation corresponding to the board shown, the true predicates would be $IsX(1, 1)$, $IsX(2, 2)$, $IsX(3, 2)$, and $IsO(1, 2)$, $IsO(2, 1)$, $IsO(3, 3)$. For convenience, we also include a unary function symbol $flip$ that reflects row or column numbers (i.e., $flip(i) = 4 - i$ for $i \in \{1, 2, 3\}$).

\(^1\)Notice that we do not explicitly address the case in which variables appear in $t_1, \ldots, t_n$. See section 2.3.
\(^2\)Strictly speaking, only free occurrences of $x$ should be replaced in this way. See section 2.3.1.
Starting configuration. The initial board configuration is described by:

$$\forall i \forall j (\neg IsX(i, j) \land \neg IsO(i, j))$$

Winning configuration. A winning configuration for the X player is one that satisfies the following formula:

$$\exists i \forall j IsX(i, j) \lor \exists j \forall i IsX(i, j) \lor \forall i IsX(i, i) \lor \forall i IsX(i, \text{flip}(i))$$

2.2.2 Example: Ancestor Relation

Note that although a predicate logic statement or collection of statements may have an intended meaning, this may or may not coincide with a particular interpretation. As an example, consider the notion of ancestor. This notion is captured by binary predicate symbols \text{IsParent} and \text{IsAncestorOf}, together with predicate logic formulas that describe the commonly understood properties of ancestry:

$$\forall x \forall y (\text{IsParentOf}(x, y) \to \text{IsAncestorOf}(x, y))$$

$$\forall x \forall y \forall z (\text{IsAncestorOf}(x, y) \land \text{IsAncestorOf}(y, z) \to \text{IsAncestorOf}(x, z))$$

The above is reasonable, and one can find a model for these statements in the usual interpretation in terms of biological ancestry. However, we’ve left some things unspecified, and so there are nonstandard models that are very different from what we intended. For example, consider the interpretation in which each individual has himself / herself as their only ancestor. In other words, the interpretation of \text{IsParentOf} and \text{IsAncestorOf} is just equality. The two formulas above would be satisfied. Clearly, we need to further describe the notion of ancestry. We can start by restricting \text{IsParentOf}:

$$\forall x \neg \text{IsParent}(x, x)$$

That doesn’t rule out a world in which there are individuals without a parent. Furthermore, we haven’t ruled out the possibility that a single individual will have many parents. Try again:

$$\text{IsParentOf}(x, y) \leftrightarrow (\text{IsFatherOf}(x, y) \lor \text{IsMotherOf}(x, y))$$

$$\forall x (\neg \text{IsFatherOf}(x, x) \land \neg \text{IsMotherOf}(x, x))$$

$$\forall x \exists y \exists z (\text{IsFatherOf}(y, x) \land \text{IsMotherOf}(z, x))$$

$$\forall x \forall y \forall z (\text{IsFatherOf}(y, x) \land \text{IsFatherOf}(z, x) \to y = z)$$

$$\forall x \forall y \forall z (\text{IsMotherOf}(y, x) \land \text{IsMotherOf}(z, x) \to y = z)$$

In any model of these statements, every individual has a unique father and a unique mother, and these are the individual’s only parents. Could something still go wrong in interpreting \text{IsAncestor} under these conditions (exercise)?
2.2.3 Example: DeMorgan-style duality for quantifiers.

\[-\forall x P \equiv \exists x \neg P\]
\[-\exists x P \equiv \forall x \neg P\]

Proof. Suppose that \( \mathcal{D} \) is a structure for which \( \mathcal{D} \models \neg \forall x P \). By the definition of truth value in a structure, this means that the truth value of \( \forall x P \) in \( \mathcal{D} \) is false, and therefore that the interpretation \( \bar{P}(d) \) does not hold for all elements \( d \in D \), where \( D \) is the domain of \( \mathcal{D} \). That in turn means that there is some element \( d^* \in D \) such that \( \bar{P}(d^*) \) does not hold, hence \( \mathcal{D} \models \exists x \neg P \). The converse is similar. It follows that \( \neg \forall x P \equiv \exists x \neg P \). To get the second formula above, apply the first one to \( \neg P \) and take the complement.

2.3 More about truth values in structures

We glossed over a very important point above when introducing the notion of truth value with respect to a structure: while terms built from constants and functions alone are easy to interpret in the domain after those building blocks have been given an interpretation, it’s not clear what to do when variables are involved. For example, consider the formula \( \text{GreaterThan}(x, 1) \) in some language that has the constant symbol 1 and the binary relation symbol \( \text{GreaterThan} \). The truth of this statement in a given structure (such as the set of integers with the usual order relation) cannot be determined unless a domain value is specified for the variable, \( x \). The situation would be different if the variable were quantified, as in \( \exists x \text{GreaterThan}(x, 1) \), as the semantics of the quantifier determine the truth value unambiguously. Thus, it is necessary to differentiate between the first type of situation, in which the variable is free, and the second, in which the variable is bound.

2.3.1 Free and bound variables

The intuitive notion of free variable is captured by the following recursive definition.

- (Basis) A variable \( x \) is free in a term \( t \) or an atomic formula \( P(t_1, \ldots, t_n) \), where \( t_1, \ldots, t_n \) are terms and \( P \) is an \( n \)-ary predicate symbol, if \( x \) appears in \( t \) (respectively, in any of \( t_1, \ldots, t_n \)).

- (Recursive step) A variable \( x \) is free in \( \phi \land \psi \), or \( \phi \lor \psi \), or \( \phi \rightarrow \psi \), or any other boolean propositional combination of formulas \( \phi \) and \( \psi \), if \( x \) is free in either \( \phi \) or \( \psi \); \( x \) is free in \( Qy \phi \), where \( Q \) is a quantifier, if \( y \) is not the same variable as \( x \) and \( x \) is free in \( \phi \).

A variable is bound when it is not free.

Example. The variable \( x \) is free in the formula below.

\[ \forall y \text{IsGreaterThan}(x, y) \]
because the outer quantified variable is not the same as \( x \) and \( x \) is free in the inner atomic formula \( \text{IsGreaterThan}(x,y) \). On the other hand, \( x \) is not free in the formula below.

\[
\forall x \text{IsGreaterThan}(x,y)
\]

However, \( x \) is free in the following formula, since it is free in the conjunct on the right.

\[
(\forall x \text{IsGreaterThan}(x,y)) \land (\exists y \text{Divides}(y,x))
\]

### 2.3.2 Variable assignments. Truth values clarified.

A variable assignment is a mapping \( \alpha : Vars \to D \) that assigns an element of the domain \( D \) to each variable in the set \( Vars \) of all variables. In other words, a variable assignment is a specialized partial interpretation in which each variable is interpreted as a specific element of the domain.

We can now clarify the notion of truth value with respect to a structure as introduced in section 2.2. The key point is that determination of the truth value of a formula requires not only a specific structure and interpretation, but also a variable assignment to at least the free variables of the formula. We will write \( \mathcal{D},\alpha \models \phi \) if \( \phi \) has the truth value \text{true} with respect to the structure \( \mathcal{D} \) and the variable assignment \( \alpha \). Another way of thinking of this is to first transform \( \phi \) by replacing all free occurrences of each variable \( x \) by the assigned value \( \alpha(x) \), yielding a formula \( \phi(x \mapsto \alpha(x)) \) that has no free variables, and then to assign a truth value to the resulting formula by using the interpretation defined over the constants, function symbols, and predicate symbols. The two approaches yield the same truth value, so that:

\[
\mathcal{D},\alpha \models \phi \text{ if, and only if, } \mathcal{D} \models \phi(x \mapsto \alpha(x))
\]

### 2.3.3 Satisfiability. Validity. Logical consequence.

A formula is called \textit{satisfiable} if it holds in some interpretation, that is, in some model, for some variable assignment to its free variables. A formula is \textit{valid} (the analog of a tautology) if it holds in \textit{all} interpretations, for all variable assignments. If \( \phi \) is valid, one writes \( \models \phi \).

If a formula \( \psi \) holds in all models in which a formula \( \phi \) holds, one writes \( \phi \models \psi \) and says that \( \psi \) is a \textit{logical consequence of} \( \phi \). Two formulas \( \phi \) and \( \psi \) are \textit{logically equivalent} (\( \phi \equiv \psi \)) if each is a logical consequence of the other.

### 2.3.4 First-order theories.

A \textit{theory} is a set \( T \) of logical formulas. A \textit{model} of \( T \) is a structure \( \mathcal{D} \) such that \( \mathcal{D} \models T \).
Example. Consider the following first-order theory over the language with constant symbols $0, 1$, variables $x, y, z$, and a single binary relation symbol $\text{LessThan}$.

$$
\forall x \neg \text{LessThan}(x, x) \\
\forall x \forall y \forall z ((\text{LessThan}(x, y) \land \text{LessThan}(y, z)) \rightarrow \text{LessThan}(x, z)) \\
\forall x (x = 0 \lor \text{LessThan}(0, x)) \\
\forall x (x = 1 \lor \text{LessThan}(x, 1))
$$

Intuitively, this theory describes all ordered structures that have distinct minimum and maximum elements. The first two formulas describe $\text{LessThan}$ as an order relation (a “partial” order). The last two formulas state that 0 is the minimum and 1 is the maximum. Two different models of this theory are shown in Fig. 1. The graphical convention is that an element $a$ is shown above $b$ with a direct edge between the two if, and only if, $\text{LessThan}(b, a)$ holds in the model.

### 3 Applications of predicate logic

#### 3.1 Loop invariants revisited

We had considered the code below to convert a list of bits to a decimal integer value.

```python
>>> def readBinary(bits):
...     n = 0
...     while len(bits) > 0:
...         n = 2*n + bits[0]
...     bits = bits[1:]
...     return n
```

The propositional loop invariant that we had arrived at for the above code is:
The numerical value $n_0$ of the original bit string may be recovered as $2^{\text{len(bits)}} n$ plus the numerical value represented by $\text{bits}$.

Although the statement is relatively clear, it is expressed mostly in natural language. Predicate logic allows us to describe the loop invariant in greater detail as follows, where $\text{bits}_0$ is the original bit string:

$$\text{actualValue}(\text{bits}_0) = \text{product}(\text{power}(2, \text{len(bits)}), n) + \text{actualValue}(\text{bits})$$

Here, the $\text{actualValue}$ function is assumed to return the correct numerical value corresponding to its bit string argument, and the $\text{product}$ and $\text{power}$ functions have the standard meanings. On the final evaluation of the loop condition, $\text{bits}$ is $[\ ]$, so that the loop invariant states:

$$\text{actualValue}(\text{bits}_0) = n + \text{actualValue}([\ ])$$

This implies the correctness of the $\text{readBinary}$ function, since the value of $n$ at that point is what is returned.

### 3.2 Graph properties

If the domain of discourse is a graph, predicate logic allows description of many properties of interest. A graph can be defined as an ordered pair $(V, E)$, where $V$ is the set of vertices and $E$ is the set of edges. We will assume that edges are undirected, so that an edge from $a$ to $b$ is the same as an edge from $b$ to $a$. Such an edge can be defined as a two-element subset of $V$, containing the two vertices that comprise the edge.

**Paths. Connectedness.** A *path* in a graph is an ordered sequence of vertices of the graph such that consecutive vertices of the path are connected by an edge of the graph.

The existence of a path between a given pair of vertices is captured by the $\text{IsConnected}$ relation, defined by recursion below:

$$\forall n \text{IsConnectedTo}(n, n)$$
$$\forall n \forall m (\text{IsEdge}(n, m) \rightarrow \text{IsConnectedTo}(n, m))$$
$$\forall n \forall m \forall p (\text{IsConnectedTo}(n, m) \land \text{IsConnectedTo}(m, p) \rightarrow \text{IsConnectedTo}(n, p))$$

The statement that the domain of discourse is connected, in the sense that there is a path between every pair of vertices, can now be expressed as follows:

$$\forall n \forall m \text{IsConnectedTo}(n, m)$$
Table 2: Sample Employee table.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Branch</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alan Turing</td>
<td>London, UK</td>
<td>logician, codebreaker</td>
</tr>
<tr>
<td>2</td>
<td>John Von Neumann</td>
<td>Princeton, NJ, USA</td>
<td>hardware architect, logician, generalist</td>
</tr>
<tr>
<td>5</td>
<td>Donald Knuth</td>
<td>Palo Alto, CA, USA</td>
<td>algorithmist, typesetter</td>
</tr>
</tbody>
</table>

3.3 Databases

Databases organize data to make it efficiently and transparently accessible. In his seminal 1970 paper in relational database theory, “A Relational Model of Data for Large Shared Data Banks”, Communications of the ACM 13 (6): 377-387, Edgar Codd of IBM proposed the idea that a database is essentially a relational structure described by first-order predicate logic. The precise syntax of languages used to describe database relations and queries (data access requests) has evolved over time, but relational database theory retains this basic viewpoint at its core. We use predicate logic notation below for consistency with the preceding discussion.

3.3.1 Relational tables

As an example, consider the notion of a database table, such as an Employee table like the one shown in Table 3.3.1. The table consists of data instances, each described by the values of one or more attributes such as id, name, address, position, and compensation. You can view such a table as a relation called Employee, consisting of all tuples that appear as instances of the table. For example, the first entry of the table can be represented as the tuple (1, 'Alan Turing', 'London, UK', 'logician, codebreaker'). The entire Employee relation in this example consists of precisely three such tuples. A relational database in this context might include the Employee table, together with additional tables such as Branches (e.g., detailed information on the 'London, UK' branch would appear as an entry in the Branches table, with two attributes called Branch and Details).

3.3.2 Relational statements

Properties of the database at a particular time can be phrased as predicate logic statements. For example, one could be interested in stating that (or determining if) there is at least one employee at each branch. This corresponds to determining the truth value of the following formula.

\[ \forall b (\exists d \text{Branch}(b, d) \rightarrow \exists i \exists n \exists r \text{Employee}(i, n, b, r)) \]

Suppose that the only branches in the database are London, Princeton, and Palo Alto. If \( \phi \) denotes the preceding formula about the presence of at least one employee at each branch,

Footnote: Codd motivated his proposal in part by the desire to insulate the user from details of the internal data representation. This motivation is similar to that behind other uses of abstraction in computer science (e.g., procedural abstraction in programming).
and if $\mathcal{D}$ denotes the database, then $\mathcal{D}$ satisfies $\phi$:

$\mathcal{D} \models \phi$

If a new branch is added in Shanghai but the Employee table is not updated, the new database $\mathcal{D}'$ would not satisfy $\phi$:

$\mathcal{D}' \not\models \phi$

### 3.3.3 Database queries

Queries are data access requests in terms of partial information (questions with information about constraints). For example, one might wish to determine all employees that have 'logician' among their responsibilities. This constraint on the responsibilities can be expressed by the following formula, $\psi$:

$$Employee(i, e, b, r) \land Contains(r, 'logician')$$

The variables $i, e, b, r$ are free in $\psi$, as one leaves open the identities of the employees that satisfy $\psi$. Answering the query entails determining all domain values of these free variables that will make $\psi$ true. Any such combination of domain values comprises a variable assignment, in the sense discussed in section 2.3.2. Therefore, answering the query reduces to finding all satisfying variable assignments for $\psi$. That is, the answer to the query represented by $\psi$ is the set of all variable assignments $\alpha$ to the free variables of $\psi$ such that $\mathcal{D}, \alpha \models \psi$.

### 4 Exercises

1. For each of the following Python functions,
   (i) Write a universally quantified predicate logic formula that you believe describes the input-output behavior of the function. Introduce constants, function symbols, and relation symbols as necessary. Explain the meaning of all language elements used.
   (ii) Prove that the formula in (i) holds. Explain in detail.

(a) >>> def g(n):
    ...     if n==0: return ''
    ...     return 'aa' + g(n-1) + 'b'

(b) >>> def f(n):
    ...     if n==0: return 0
    ...     return 3 + 2*n + f(n-1)

2. Consider the language $L$ of all first-order predicate logic formulas that allow the constant 0, the two variables $x$ and $y$, the single binary predicate symbol $P$, and the single unary function symbol $f$. Find the smallest set $S$ such that $L \subseteq S^*$ (Kleene closure). Use the formal definition of predicate logic syntax in these notes. Explain in detail.
3. Write down a detailed recursive definition of interpretation with respect to a structure for general predicate logic terms.

4. Describe two specific models of each of the following first-order theories. Explain.

   (a) 
   $$\forall x \neg \text{LessThan}(x, x)$$
   $$\forall x \forall y \forall z ((\text{LessThan}(x, y) \land \text{LessThan}(y, z)) \rightarrow \text{LessThan}(x, z))$$
   $$\forall x (x = 0 \lor \text{LessThan}(0, x))$$
   $$\forall x \exists y \exists z (-(y = z) \land \text{LessThan}(x, y) \land \text{LessThan}(x, z) \land$$
   $$\land \forall w (\text{LessThan}(x, w) \rightarrow (y = w \lor z = w \lor \text{LessThan}(y, w) \lor \text{LessThan}(z, w))))$$

   (b) 
   $$\forall x \neg \text{LessThan}(x, x)$$
   $$\forall x \forall y \forall z ((\text{LessThan}(x, y) \land \text{LessThan}(y, z)) \rightarrow \text{LessThan}(x, z))$$
   $$\forall x \forall y (x = y \lor \text{LessThan}(x, y) \lor \text{LessThan}(y, x))$$
   $$\forall x (x = 0 \lor \text{LessThan}(0, x))$$
   $$\forall x \exists y (\text{LessThan}(x, y) \land \forall z (\text{LessThan}(x, z) \rightarrow (z = y \lor \text{LessThan}(y, z))))$$

5. Fix the remaining problems in the ancestry definition in section 2.2.2, so that the models of the resulting theory will be consistent with the standard notion of ancestry.

6. For each of the following formulas,
   (i) List the free variables of $\phi$.
   (ii) Find all assignments of the free variables that will make the given formula true in the given structure.

   (a) The formula 
   $$\forall x \text{Divides}(x, y) \rightarrow x = 1 \lor x = y$$
   in the structure 
   $$\{n | n \text{ is an integer } \geq 0\}$$
   with the usual interpretation for Divides and 1.

   (b) The formula 
   $$\phi : \neg \text{LessThan}(x, y) \land \neg \text{LessThan}(y, x)$$
   in the structure 
   $$\mathcal{D} = \{S | S \subseteq \{1, 2, 3\}\}$$
   with LessThan interpreted as strict subset inclusion (e.g., LessThan(\{a\}, \{a, b\}) would hold, but LessThan(\{a, b\}, \{a, b\}) would not).