Declarative (logic) programming uses a form of symbolic logic as a programming language. In this lecture, we introduce Prolog, a widely known declarative programming language. Prolog programs consist of a collection of statements. Each statement is constructed from terms.

A term is a constant, a variable, or a structure.

A constant is either an atom or an integer. An atom is a symbolic value, which is either:
- A string of letters, digits, and underscores that begins with a lowercase letter, or:
- A string of characters delimited by single quotes.

A variable is any string of letters, digits, and underscores that begins with an uppercase letter. A variable does not bind to type by declaration. It binds dynamically to a type when it is assigned a value. Such binding is called instantiation.

A structure consists of an atom (called functor) followed by a parameter list of terms inside ().

There are three types of statements in Prolog: fact statements, rule statements, and goal statements.

A fact statement consists of a structure followed by dot. Fact statements are propositions that are assumed to be true. Consider the following examples of fact statements:

| male(bill). |
| female(ann). |
| father(bill, ann). |

Such facts have no intrinsic meaning. They mean whatever the programmer wants them to mean. Here, we assume that male(bill) means that bill is male. Also, father(bill, ann) means that bill is the father of ann, and so on.

Rule statements are mechanisms to conclude new facts from given facts. For example, the rule:

```
parent(X, Y) :- mother(X, Y). // If RHS of :- is true then LHS is true
```

means that for any X and Y: If X is mother of Y, then X is parent of Y. Also, the rule:

```
grandmother(X, Z) :- mother(X, Y), parent(Y, Z). // Comma = AND
```

means for any X, Y, Z: If X is mother of Y, and Y is parent of Z, then X is grandmother of Z.

The right hand side (RHS) of a rule statement (the part after :-) is the antecedent (the if part). The left hand side (LHS) is the consequent (the then part). If the antecedent of a rule statement is true, then its consequent must be true.

The consequent is a single term, while the antecedent can be either a single term or a conjunction. Conjunctions contain multiple terms separated by logical AND operations implied by commas.
Consider the following fact statements:

| male(jake) | male(bill) |
| female(ann) | female(mary) |
| father(bill, jake) | father(bill, ann) |
| mother(mary, jake) | mother(mary, ann) |

Also, consider the following rule statements:

\[
\begin{align*}
\text{parent}(X, Y) & : - \text{mother}(X, Y). \\
\text{parent}(X, Y) & : - \text{father}(X, Y). \\
\text{grandparent}(X, Z) & : - \text{parent}(X, Y), \text{parent}(Y, Z).
\end{align*}
\]

Fact and rule statements are the basis for the theorem proving model. A goal statement (or query) is a proposition that we want the system to either prove or disprove. Its syntax is similar to a fact statement: a structure followed by dot. But it is not part of the database, it is just a query that needs to be checked against the existing database of facts and rules.

Consider the following goal (query):

male(bill).

The system will try to prove the goal given the database of facts and rules. It will find a match and then will output yes, which means that the goal is true.

Consider the following goal:

male(john).

The system will try to prove the goal given the database of facts and rules. It will not find a match and then will output no, which means that the goal cannot be proved given the existing database. It does not necessarily mean that the goal is false.

Consider the following goal:

male(X).

Since X is a variable (because its initial letter is uppercase), it matches fact: male(jake). A variable can match any term. Thus, the system outputs X=jake, which implies yes. Also, it matches fact: male(bill) and the system also outputs X=bill. An uninstantiated variable can match with any constant or structure. Similarly, for the goal:

father(bill, X).

The system outputs X=jake and X=ann. Similarly, for the goal:

mother(X, jake).

The system outputs X=mary. Similarly, for the goal:

father(X, Y).

The system outputs X=bill, Y=jake and X=bill, Y=ann.
Consider the following **conjunctive goal**:

\[
\text{father}(X, Y), \text{female}(Y).
\]

This **goal** is composed of two **subgoals** that both need to be matched simultaneously. The system first attempts to match with the **fact**: \text{father(bill, jake)} so it sets \(X=bill, Y=jake\). Now, all subsequent **subgoals** must be matched without changing the values of \(X\) and \(Y\). So, the system attempts to match \text{female(jake)} which is not possible. Thus, the system concludes that it did not reach the goal with such \(X\) and \(Y\) **instantiations**.

Hence, the system **backtracks** and **uninstantiates** \(X\) and \(Y\), then it attempts to match the first **subgoal** with another **fact** which is: \text{father(bill, ann)} so it sets \(X=bill, Y=ann\). Now, the system attempts to match \text{female(ann)} which is possible, and then it outputs \(X=bill, Y=ann\). Note that the system would have **backtracked** also even if the first instantiation was successfully matched, in order to get all possible solutions. The system will **backtrack** now also trying to find other solutions but it could not.

An important note about the **inferencing process** of **Prolog** is that the value of a **variable** can change only after the system **backtracks** and **uninstantiates** all variables that have been **instantiated** after it. Otherwise, the value of a **variable** cannot change. This is the main difference between **procedural** and **declarative programming**, where a variable in **procedural programming** simulates a **memory cell**, while a variable in **declarative programming** simulates a **mathematical variable**.

The above **queries** involve **facts** only. Now we consider more complex **queries** which involve **rules** as well. Consider the following **goal**:

\[
\text{parent}(Y, \text{ann}).
\]

The system **matches** \text{parent}(Y, \text{ann}) with **LHS** of rule \text{parent}(X, Y):\neg\text{mother}(X, Y). Note that there are two different variables with the same name \(Y\) which occur in different contexts. \(Y\text{(goal)}\) **matches** \(X\) and \(\text{ann}\) **matches** \(Y\text{(rule)}\). “\(Y\text{(goal)}\) **matches** \(X\)” means that if one of \(X\) or \(Y\) **instantiates** to a value, the other variable will **instantiate** to the same value. “\(\text{ann}\) **matches** \(Y\text{(rule)}\)” means that \(Y\text{(rule)}\) **instantiates** to the value \(\text{ann}\).

Now, the system attempts to **match** the **RHS** of the rule, which is \text{mother}(Y\text{(goal)}, \text{ann}), because if the **RHS** of the rule is proved, the **LHS** of this rule is implied and proved as well. It **matches** \text{mother}(\text{mary}, \text{ann}) and the system outputs \(Y=\text{mary}\).

Then, the system **backtracks** trying to find other solutions. It **uninstantiates** \(Y\text{(goal)}\) and **matches** \text{parent}(Y, \text{ann}) with **LHS** of rule \text{parent}(X, Y):\neg\text{father}(X, Y). \(Y\text{(goal)}\) **matches** \(X\) and \(\text{ann}\) **matches** \(Y\text{(rule)}\). The system attempts to **match** \text{father}(Y\text{(goal)}, \text{ann}). It succeeds to **match** it with \text{father}(\text{bill}, \text{ann}) and the system outputs \(Y=\text{bill}\).

Consider the following **conjunctive goal**:

\[
\text{parent}(Y, \text{ann}), \text{male}(Y).
\]

The system attempts to **match** the first **subgoal** \text{parent}(Y, \text{ann}) and gets a solution \(Y=\text{mary}\), then it attempts to **match** the second **subgoal** \text{male(\text{mary})} but it does not succeed.

Then, the system **backtracks** and gets another solution to \text{parent}(Y, \text{ann}) which is \(Y=\text{bill}\), then attempts to match the second **subgoal** \text{male(\text{bill})} and it succeeds, so it outputs \(Y=\text{bill}\).
Consider the following goal:

\[
\text{grandparent(bill, mary).}
\]

The system matches \text{grandparent(bill,mary)} with \text{LHS} of rule \text{grandparent(X,Z):-parent(X,Y),parent(Y,Z).} So, \text{bill matches X}, and \text{mary matches Z.}

Now, the system attempts to match both \text{parent(bill,Y), parent(Y,mary)} because all \text{subgoals} of the \text{RHS} of a rule are required to be proved in order to prove the \text{LHS} of this rule.

The first \text{subgoal parent(bill,Y)} returns the solution \text{Y=jake}, then the system attempts to match the second \text{subgoal parent(jake,mary)} but it does not succeed.

The system backtracks and finds another solution to the first \text{subgoal parent(bill,Y)} which is \text{Y=ann}, then the system attempts to match \text{parent(ann,mary)} but it does not succeed.

Then, the system backtracks trying to match \text{grandparent(bill,mary)} with the \text{LHS} of another rule but it does not find another rule. Hence the system decides that it cannot prove the main \text{goal} and outputs \text{no}.

The above inferencing process is called \text{top-down resolution} (or \text{backward chaining}) because it starts from the \text{goal} (or \text{subgoals}) and attempts to find a sequence of \text{matching} propositions that lead to some set of original \text{facts} in the \text{database}. This approach works well when there is a reasonably small set of candidate answers.

An alternative method for the inferencing process is \text{bottom-up resolution} (or \text{forward chaining}) which begins with the \text{facts} and \text{rules} of the database and attempts to find a sequence of \text{matches} that lead to the \text{goal}. This method is not used in \text{Prolog}, but it is usually better when the number of possibly correct answers is large.