

The input query specifies that we are looking for entries in the data base that match a certain pattern.


The query may have zero or more variables.

A query can have no variables, in which case the query simply determines whether that pattern is a fact in the data base or can be inferred from the facts and with the help of the rules stored in the database.


```

boss(robert,teja).
boss(bhalla,robert).
boss(mogambo,mogambo).

```

 `boss(X,X).`

**X** = mogambo

 `boss(X,Y).`

**X** = robert,

**Y** = teja

**X** = bhalla,

**Y** = robert

**X** = Y, **Y** = mogambo

If a pattern has more than one variables, then the variables maybe same or different.

For example,

`boss(X,Y)` will return all three facts, whereas the `boss(X,X)` will only return the last fact in the database as the query specifies the two atoms to be same.

We can describe the query language's processing of simple queries as follows:

- ▶ The system finds all assignments to variables in the query pattern that satisfy the pattern—that is, all sets of values for the variables such that if the pattern variables are *instantiated with* (replaced by) the values, the result is in the data base.
- ▶ The system responds to the query by listing all instantiations of the query pattern with the variable assignments that satisfy it.

If the pattern has no variables, the query reduces to a determination of whether that pattern is in the data base.

Logic programming combines a relational vision of programming with a powerful kind of symbolic pattern matching called **unification**.

A pattern matcher is a program that tests whether some datum fits a specified pattern.

For example, the fact  $((a\ b)\ c\ (a\ b))$  matches the pattern  $(X\ c\ X)$  with the pattern variable  $X$  bound to  $(a\ b)$ .

However, it does not match the pattern  $(X\ a\ Y)$ , since that pattern specifies a list whose second element is the symbol  $a$ .

- ▶ The pattern matcher used by the query system takes as inputs a pattern, a datum, and a frame that specifies bindings for various pattern variables.

- ▶ The pattern matcher used by the query system takes as inputs a pattern, a datum, and a frame that specifies bindings for various pattern variables.
- ▶ It checks whether the datum matches the pattern in a way that is consistent with the bindings already in the frame.

- ▶ The pattern matcher used by the query system takes as inputs a pattern, a datum, and a frame that specifies bindings for various pattern variables.
- ▶ It checks whether the datum matches the pattern in a way that is consistent with the bindings already in the frame.
- ▶ If so, it returns the given frame augmented by any bindings that may have been determined by the match. Otherwise, it indicates that the match has failed.

For example, using the pattern  $(X\ Y\ X)$  to match  $(a\ b\ a)$

1. given an empty frame will return  
a frame specifying that  $X$  is bound to  $a$  and  $Y$  is bound to  $b$ .



For example, using the pattern  $(X\ Y\ X)$  to match  $(a\ b\ a)$

1. given an empty frame will return  
a frame specifying that  $X$  is bound to  $a$  and  $Y$  is bound to  $b$ .
2. given a frame specifying that  $Y$  is bound to  $a$  will fail.

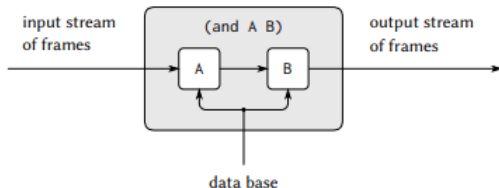
For example, using the pattern  $(X\ Y\ X)$  to match  $(a\ b\ a)$

1. given an empty frame will return  
a frame specifying that  $X$  is bound to  $a$  and  $Y$  is bound to  $b$ .
2. given a frame specifying that  $Y$  is bound to  $a$  will fail.
3. given a frame in which  $Y$  is bound to  $b$  and  $X$  is unbound will  
return the given frame augmented by a binding of  $X$  to  $a$ .

For example, using the pattern  $(X\ Y\ X)$  to match  $(a\ b\ a)$

1. given an empty frame will return  
a frame specifying that  $X$  is bound to  $a$  and  $Y$  is bound to  $b$ .
2. given a frame specifying that  $Y$  is bound to  $a$  will fail.
3. given a frame in which  $Y$  is bound to  $b$  and  $X$  is unbound will  
return the given frame augmented by a binding of  $X$  to  $a$ .

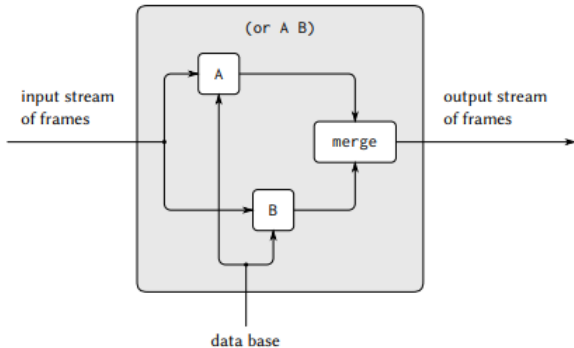
The pattern matcher is all the mechanism that is needed to process simple queries that don't involve rules.



The *and* of two queries can be viewed as a series combination of the two component queries.

The frames that pass through the first query filter are filtered and further extended by the second query.

The *or* of two queries can be viewed as a parallel combination of the two component queries. The input stream of frames is extended separately by each query. The two resulting streams are then merged to produce the final output stream.



From the stream-of-frames viewpoint, the not of some query acts as a filter that removes all frames for which the query can be satisfied.

For instance, given the pattern `not (job(X (computer programmer)))` we attempt, for each frame in the input stream, to produce extension frames that satisfy `job (X (computer programmer))`.

Then we remove from the input stream all frames for which such extensions exist.

Rule conclusions are like assertions except that they can contain variables, so we will need a generalization of pattern matching called unification in which both the “pattern” and the “datum” may contain variables.

A unifier takes two patterns, each containing constants and variables, and determines whether it is possible to assign values to the variables that will make the two patterns equal. If so, it returns a frame containing these bindings.

For example, unifying  $(X\ a\ Y)$  and  $(Y\ Z\ a)$  will specify a frame in which  $X$ ,  $Y$ , and  $Z$  must all be bound to  $a$ .

On the other hand, unifying  $(X\ Y\ a)$  and  $(X\ b\ Y)$  will fail, because there is no value for  $Y$  that can make the two patterns equal.

The unifier used in the query system, like the pattern matcher, takes a frame as input and performs unifications that are consistent with this frame.



The unification algorithm is the most technically difficult part of the query system. With complex patterns, performing unification may seem to require deduction.

In a successful pattern match, all pattern variables become bound, and the values to which they are bound contain only constants.

In general, however, a successful unification may not completely determine the variable values; some variables may remain unbound and others may be bound to values that contain variables.

Consider the unification of  $(X\ a)$  and  $((b\ Y)\ Z)$ .

We can deduce that  $X = (b\ Y)$  and  $a = Z$ , but we cannot further solve for  $X$  or  $Y$ .

Consider the unification of  $(X\ a)$  and  $((b\ Y)\ Z)$ .

We can deduce that  $X = (b\ Y)$  and  $a = Z$ , but we cannot further solve for  $X$  or  $Y$ .

The unification doesn't fail, since it is certainly possible to make the two patterns equal by assigning values to  $X$  and  $Y$ .

Since this match in no way restricts the values  $Y$  can take on, no binding for  $Y$  is put into the result frame. The match does, however, restrict the value of  $X$ . Whatever value  $Y$  has,  $X$  must be  $(b\ Y)$ .

Another way to think of unification is that it generates the most general pattern.