Advanced Logic and Functional Programming

Lecture 2: Logic Programming in Prolog

Mircea Marin

october 2018

Organizatorial items

- Lecture & labs: Mircea Marin
  - website: `http://staff.fmi.uvt.ro/~mircea.marin/lectures/ALFP/`
  - email: `mircea.marin@e-uvt.ro`
- The slides of lecture notes and the labworks will be posted on the website.
- Evaluation:
  - one written exam (50% of the final grade)
  - Labworks & miniproject (50% of the final grade).

  Lab attendance is mandatory.

What is logic programming?

Programming style where

- Program = collection of rules and facts which represent the knowledge base of the programmer for the problem he wants to solve.
- Program execution = finding answers to questions specified by the programmer.
    - The answers are found by a process of logical reasoning based on the knowledge stored in the user program.

What is logic programming?

Programming style where

- Program = collection of rules and facts which represent the knowledge base of the programmer for the problem he wants to solve.
- Program execution = finding answers to questions specified by the programmer.
  - The answers are found by a process of logical reasoning based on the knowledge stored in the user program.

▷ A logic programmer **must know**

- how to encode his knowledge about the problem as a collection of facts and rules
- how to encode questions (a.k.a. queries) in Prolog

What is logic programming?

Programming style where

- Program = collection of rules and facts which represent the knowledge base of the programmer for the problem he wants to solve.
- Program execution = finding answers to questions specified by the programmer.
  - The answers are found by a process of logical reasoning based on the knowledge stored in the user program.

▷ A logic programmer **must know**

- how to encode his knowledge about the problem as a collection of facts and rules
- how to encode questions (a.k.a. queries) in Prolog

▷ The interpreter (or compiler) of PROLOG **knows** how to find all correct answers to Prolog questions.

Characteristics of Logic Programming

Logic programming is a declarative programming style

- Programmer must know how to write a program which described what he knows about the problem he wants to solve.
- The method how to find the answers to questions is predefined, and used by the Prolog interpreter or compiler.
  - PROLOG uses a strategy to find all answers to questions, which is based on logical reasoning: It uses a rule of deduction called SLD resolution.

Facts

Facts are atomic statements about objects and the relations which exist between them.

- Facts of type *property*(*object*):

  Coco is a parrot: `parrot(coco).`

  Every (X) is mortal: `mortal(X).`

- Facts can express relations which exist among objects:

  Mike loves Mary: `loves(mike,mary).`

  Every (X) is equal to itself: `equal(X,X).`

Rules

- In natural language, a rule is a sentence of the form:
  - **If** *hypothesis*$_1$ and ... and *hypothesis*$_n$ **then** *conclusion*.

  $\Leftrightarrow$ *conclusion* **if** *hypothesis*$_1$ and ... and *hypothesis*$_n$.

  where *hypothesis*$_1$, ..., *hypothesis*$_n$ and *conclusion* are atomic statements.
- In Prolog, the rule is written as follows:

$$\textit{conclusion} :- \textit{hypothesis}_1, \ldots, \textit{hypothesis}_n.$$

Rules

- In natural language, a rule is a sentence of the form:
  - **If** *hypothesis*$_1$ and ... and *hypothesis*$_n$ **then** *conclusion*.

  $\Leftrightarrow$ *conclusion* **if** *hypothesis*$_1$ and ... and *hypothesis*$_n$.

  where *hypothesis*$_1$, ..., *hypothesis*$_n$ and *conclusion* are atomic statements.
- In Prolog, the rule is written as follows:

$$\textit{conclusion} :- \textit{hypothesis}_1, \ldots, \textit{hypothesis}_n.$$

Example

In natural language (English)	**În PROLOG**
If X is good and X knows Y and Y is pretty then X loves Y.	loves(X, Y) :– good(X), knows(X, Y), pretty(Y).
Every parrot is mortal. (If X is parrot then X is mortal.)	mortal(X) :–parrot(X).

Queries (or questions)

In natural language (English)	In PROLOG
Is Coco a parrot?	?-parrot(coco).
Who is mortal? (For what values of X do we know that X is mortal?)	?-mortal(X).

- in logic programming, we solve problems by asking questions of the following kinds:
  - " Is it true that . . . ?"
  - "For what values of the variables . . . is true that . . . ?"
- The programmer **need not know how** to find answers to queries: this task belongs to the interpreter of Prolog.

From natural language to Prolog

- knowledge written in a natural language must be encoded as rules and facts in Prolog.
- Often, it is useful to rephrase the sentences in natural language, to simplify the translation process in PROLOG.

Example

- In natural language:
  Hardworking students take good grades. $\Leftrightarrow$ If X is student and X is hardworking then X takes good grades $\Leftrightarrow$ X takes good grades if X is student and X is hardworking.
- In artificial language (PROLOG):

$$\text{grades}(X, \textit{good}) :- \text{student}(X), \text{hardworking}(X).$$

REMARK: n PROLOG, $:-$ means "if" and the comma between hypotheses (,) is "and".

Logic Programming: Short History

Kowalski (in 70s) noticed that a logical formula

$$S_1 \wedge \ldots \wedge S_n \rightarrow S$$

can be interpreted in two ways:

- logic interpretation: If S_1 and ... and S_n are all true then S is also true.
- procedural interpretation of "S if S_1 and ... and S_n" is: in order to find out if S is true we must check recursively if S_1, ..., S_n are true.

În PROLOG, this formula becomes a rule

$$S :- S_1, \ldots, S_n.$$

where S is the head of the rule, and $S_1, \ldots, S_n$ constitute the body of the rule.

Logic Programming: Short History (contd.)

I

- University of Marsilia (Colmerauer, in 70s): language PROLOG appears ("**Pro**grammation et **Log**ique".)
- PROLOG became the most popular language of logic programming
  - ⇒ currently, there are several interpreters and compilers available

Solving problems with PROLOG

- We declare the facts we know about objects and the relations between them.

Solving problems with PROLOG

- We declare the facts we know about objects and the relations between them.
- We declare the rules we know about objects and the relations between them.

Solving problems with PROLOG

- We declare the facts we know about objects and the relations between them.
- We declare the rules we know about objects and the relations between them.
- We ask questions about objects and the relations between them.

Solving problems with PROLOG

- We declare the facts we know about objects and the relations between them.
- We declare the rules we know about objects and the relations between them.
- We ask questions about objects and the relations between them.

Programming in PROLOG is a dialog with the interpreter. Solving problems in this way requires to know how to model the problem using the notions of logic programming:

- facts.
- rules.
- queries.

$$\text{Program} = \underbrace{\text{facts} + \text{rules}}_{\text{knowledge base}}.$$

Facts

- In PROLOG, facts are specified as follows

$$\textit{predicate}(\underbrace{\textit{object}_1, \ldots, \textit{object}_n}_{\text{arguments}}).$$

  For example

$$\textit{has}(\textit{andrew}, \textit{book}).$$

  - the names of relations (predicates) start with lowercase letter.
  - Usually, PROLOG uses prefix notation to specify relations, but there are exceptions too.
  - Every fact ends with a "." (dot).
- The programmer is free to choose predicate names, and to decide how to interpret them.
  - For example., $\textit{has}(\textit{andrew}, \textit{book}).$ means *Andrew has a book*.

Examples of facts

- Gold is precious.
  `precious(gold).`
- Jane is a woman.
  `woman(jane).`
- Jon is Mary's father.
  `father(jon, mary).`
- Andrew has a book.
  `has(andrew, book).`

Remarks

- The programmer must fix the meanings of the names of objects and predicates predicates he uses.
- For example: `has(X,Y)` means X has Y, which is different from Y has X.
  The order of arguments matters!

- Example of query in PROLOG

?– has(andrew, book).

(Does Andrew have a book?)

Quesries

- Example of query in PROLOG

  ?− has(andrew, book).

  (Does Andrew have a book?)
  PROLOG consults the program(=knowledge base) to find facts which match the questions.

Quesries

- Example of query in PROLOG

  ?- has(andrew, book).

  (Does Andrew have a book?)
  PROLOG consults the program(=knowledge base) to find facts which match the questions.
- The answer is `true` if
  - the predicate is the same
  - the arguments are the same

Quesries

- Example of query in PROLOG

  ?− has(andrew, book).

  (Does Andrew have a book?)
  PROLOG consults the program(=knowledge base) to find facts which match the questions.
- The answer is `true` if
  - the predicate is the same
  - the arguments are the same
- Otherwise, the answer is `false`

Quesries

- Example of query in PROLOG

  ?– has(andrew, book).

  (Does Andrew have a book?)
  PROLOG consults the program(=knowledge base) to find facts which match the questions.
- The answer is `true` if
  - the predicate is the same
  - the arguments are the same
- Otherwise, the answer is `false`
  - Only what can be found in the knowledge base is assumed to be true.

Quesries

- Example of query in PROLOG

  `?-` has(andrew, book).

  (Does Andrew have a book?)
  PROLOG consults the program(=knowledge base) to find facts which match the questions.
- The answer is `true` if
  - the predicate is the same
  - the arguments are the same
- Otherwise, the answer is `false`
  - Only what can be found in the knowledge base is assumed to be true.
  - **The `false` answer is not the same with false!**
    - Answer `false` means I don't know, and `true` means I know.

Variables

Variable = placeholder for an object that satisfies a relation.

- Example of a question with a variable:

  ?− likes(jon, X).

  is interpreted as follows:
  - Which are the objects *X* that Jon likes?

  PROLOG will start to look for the values of *X* for which the answer is `true`.
- **Convention**: In PROLOG variables start with _ or with uppercase letter.
- In PROLOG a variable can be
  - instantiated: the variable has an object as its value
  - uninstantiated: we don't know yet a value for that variable.

Example of query with variables

- Consider the following program:

  likes(jon,flowers).
  likes(jon,mary).
  likes(paul,mary).

- To the query

  ?−likes(jon, X).

  PROLOG answers

  X = flowers

  and will wait for further instructions from the user.

How does PROLOG find alswers?

- PROLOG looks in the program for a fact which matches the query
- when a match is found, the place of the match is marked
- if the user clicks `Enter`, the search for answers stops.
- if the user clicks ";", PROLOG looks for another match, starting from the last marked place, and with the variables to the query uninstantiated.
- In the previous example, if we press ";", PROLOG will also find the nswer

  X = mary .
- When PROLOG can not find other answers in the program, it returns answer `false`

- Consider the following program:

```
likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).
```

  and the query

```
?- likes(jon, mary), likes(mary, jon).
```

- Consider the following program:

```
likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).
```

  and the query

```
?- likes(jon, mary), likes(mary, jon).
```

- In general, a query

  ?- $fact_1, \ldots, fact_n$.

  has the intended reading $fact_1$ **and** ... **and** $fact_n$?
  In this case:

  Does Jon like Mary **and** does Mary like Jon?

Knowledge base:

```
likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).
```

Query:

```
?- likes(jon, mary), likes(mary, jon).
```

PROLOG answers `false` : it looks for **all** facts in the query, from left to right (all must be satisfied, otherwise the query fails and the answer will be `false`).

Knowledge base:

```
likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).
```

Question:

```
?- likes(mary, X), likes(jon, X).
```

More complex queries (contd.)
Conjunctions

Knowledge base:

```
likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).
```

Question:

```
?- likes(mary, X), likes(jon, X).
```

- The question is: Is there an *X* who is liked by Jon and by Mary?

Knowledge base:

```
likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).
```

Question:

```
?- likes(mary, X), likes(jon, X).
```

- The question is: Is there an *X* who is liked by Jon and by Mary?
- PROLOG tries to satisfy the first sub-question; if it succeeds, it marks it and tries to satisfy the second sub-question; it it succeeds, it marks it too.

Conjunctions and backracking

```
?- likes(mary, X), likes(jon, X).
```

```
likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).
```

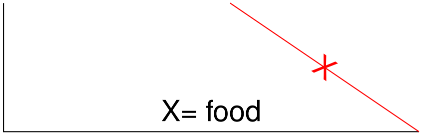
Conjunctions and backracking

?- likes(mary, X), likes(jon, X).

X= food

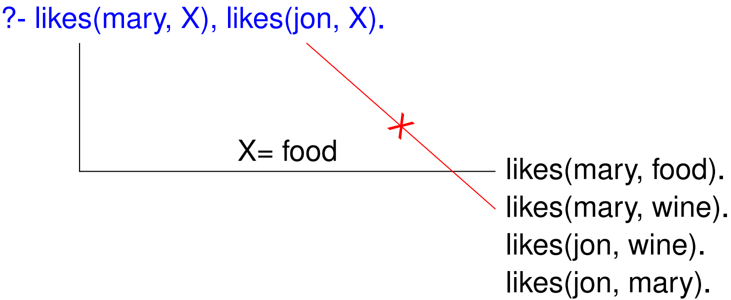
likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).

?- likes(mary, X), likes(jon, X).



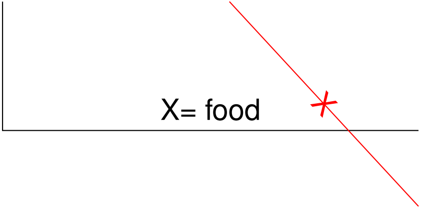
likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).

Conjunctions and backracking



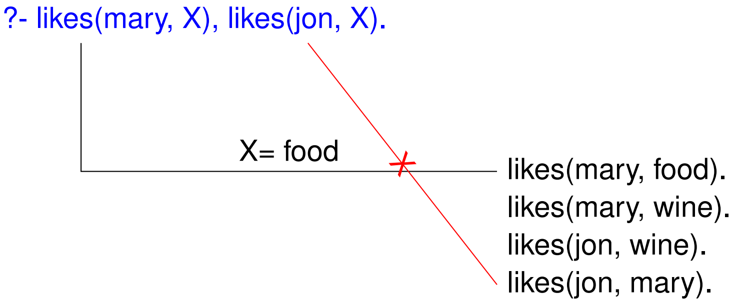
Conjunctions and backracking

?- likes(mary, X), likes(jon, X).



likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).

Conjunctions and backracking

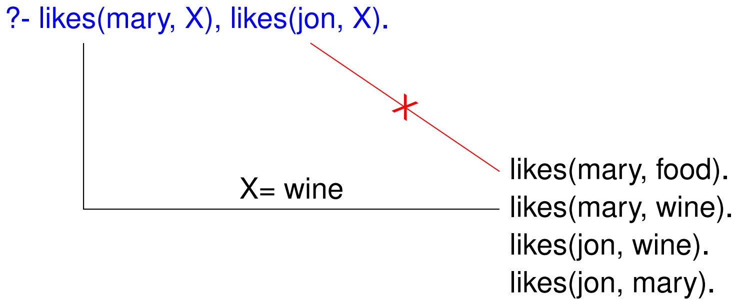


Conjunctions and backracking

?- likes(mary, X), likes(jon, X).

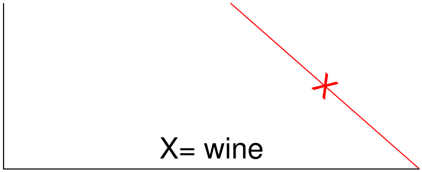
X= wine

likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).


?- likes(mary, X), likes(jon, X).
X= wine
likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).

Conjunctions and backracking

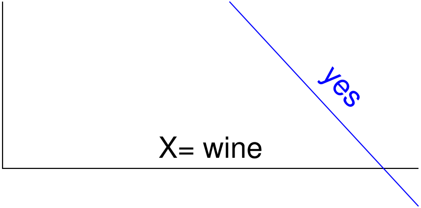
?- likes(mary, X), likes(jon, X).



likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).

Conjunctions and backracking

?- likes(mary, X), likes(jon, X).



likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).

X = wine

Conjunctions and backracking

?- likes(mary, X), likes(jon, X).

X= wine

likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).

X = wine

Conjunctions and backracking

?- likes(mary, X), likes(jon, X).

likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).

X = wine

Conjunctions and backracking

?- likes(mary, X), likes(jon, X).

X

X = wine

likes(mary, food).
likes(mary, wine).
likes(jon, wine).
likes(jon, mary).

Rules

- How can we encode the fact that "Jon likes everybody"?

  likes(jon,alex).
  likes(jon, bogdan).
  likes(jon, clara).
  likes(jon, dan).
  ...

  In this way, we should enumerate all people in the knowledge base → impossible!

Rules

- How can we encode the fact that "Jon likes everybody"?

  likes(jon,alex).
  likes(jon, bogdan).
  likes(jon, clara).
  likes(jon, dan).
  ...

  In this way, we should enumerate all people in the knowledge base $\rightarrow$ impossible!

- We can use a variabile:

  likes(jon, X).

  means "Jon likes every X."

Rules

- How can we encode the fact that "Jon likes everybody"?

  likes(jon,alex).
  likes(jon, bogdan).
  likes(jon, clara).
  likes(jon, dan).
  ...

  In this way, we should enumerate all people in the knowledge base $\rightarrow$ impossible!
- We can use a variabile:

  likes(jon, X).

  means "Jon likes every X."
- We should also specify that Jon likes every X which is a person.

Rules

- How can we encode the fact that "Jon likes everybody"?

  ```
  likes(jon,alex).
  likes(jon, bogdan).
  likes(jon, clara).
  likes(jon, dan).
  ...
  ```

  In this way, we should enumerate all people in the knowledge base $\rightarrow$ impossible!
- We can use a variabile:

  ```
  likes(jon, X).
  ```

  means "Jon likes every X."
- We should also specify that Jon likes every X which is a person.
- Rules are used to specify how one fact depends on other facts.

Rules and definitions

- **Rules can be used to specify definitions.**

Rules and definitions

- **Rules can be used to specify definitions.**
- Examples:

Rules and definitions

- **Rules can be used to specify definitions.**
- Examples:
- "*X* is liked by Jon **if** *X* is man."

Rules and definitions

- **Rules can be used to specify definitions.**
- Examples:
- "*X* is liked by Jon **if** *X* is man."
- "*X* is bird **if** *X* is animal and *X* flies."

Rules and definitions

- **Rules can be used to specify definitions.**
- Examples:
- "X is liked by Jon **if** X is man."
- "X is bird **if** X is animal and X flies."
- "X is the sister of Y **if** X is woman and X and Y have same parents."

Rules and definitions

- **Rules can be used to specify definitions.**
- Examples:
- "*X* is liked by Jon **if** *X* is man."
- "*X* is bird **if** *X* is animal and *X* flies."
- "*X* is the sister of *Y* **if** *X* is woman and *X* and *Y* have same parents."

REMARK: Rules are not the same as definitions!

- A definition says that something holds if and only if the body of the definition holds.
- A rule says that a fact (the head of the rule) holds if the body of the rule holds. It may be case that the head of the rule holds in other situations too.
  - “*X* is human **if** *X* is female.”
  - “X is human **if** *X* male.”

Rules in PROLOG

- Rules in PROLOG have a head and a body.
- The body of the rule describes conditions which, if they hold, then the fact in the head holds too.

Example

```
likes(jon, X) :-
        likes(X,wine).
likes(jon, X) :-
        woman(X), likes(X, dance).
```

ATENTION! The scope of a variable is the rule where it appears (different rules have no variables in common).

Rules in PROLOG

- Syntax of a rule

$$\overbrace{predicate(arg_1, \ldots, arg_n)}^{\text{head}} \;{:}{-}\; \overbrace{fact_1, \ldots, fact_m}^{\text{body}}\,.$$

fact delimiters (and)

rule delimiter (if)

Predicates that are used:

male(X): "X is male."
female(Y): "Y is female."
parents(X,Y,Z): "The parents of X are Y and Z."

Predicates that are used:

- male(X): "X is male."
- female(Y): "Y is female."
- parents(X,Y,Z): "The parents of X are Y and Z."

Knowledge base:

```
male(albert).
male(edward).
female(alice).
female(victoria).
parents(edward,victoria,albert).
parents(alice,victoria,albert).
```

Predicates that are used:

male(X): "X is male."
female(Y): "Y is female."
parents(X,Y,Z): "The parents of X are Y and Z."

Knowledge base:

```
male(albert).
male(edward).
female(alice).
female(victoria).
parents(edward,victoria,albert).
parents(alice,victoria,albert).
```

- How can we define the predicate
  sister(X, Y): "X is the sister of Y."?

- Definition of the predicate sister/2:

```
sister(X,Y) :-
        female(X),
        parents(X, F, M),
        parents(Y, F, M).
```

Examples of queries:

```
?- sister(alice, edward).
?- sister(alice, X).
?- sister(X, edward).
```

Questions about sisters

Rule:

```
sister(X,Y) :-  female(X),
                parents(X,F,M),
                parents(Y,F,M).
```

Question:

```
sister(alice,edward).
```

Questions about sisters

Rule:

```
sister(X,Y) :-  female(X),
                parents(X,F,M),
                parents(Y,F,M).
```

Question:

```
sister(alice,edward).
```

- The query matches the head of the rule. The matching instantiates `X` with `alice` and `Y` with `edward`.

Questions about sisters

Rule:

```
sister(X,Y) :-  female(X),
                parents(X,F,M),
                parents(Y,F,M).
```

Question:

```
sister(alice,edward).
```

- The query matches the head of the rule. The matching instantiates `X` with `alice` and `Y` with `edward`.
- The body of the rule is instantiated and becomes the new query:

```
    female(alice),
    parents(alice,F,),
    parents(edward,F,M).
```

Is Alice the sister of Edward?

```
sister(alice,edward)
```

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

Is Alice the sister of Edward?

`sister(alice,edward)`

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

Is Alice the sister of Edward?

sister(alice,edward)

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

Is Alice the sister of Edward?

`sister(alice,edward)`

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X0,Y0):-
      female(X0),
      parents(X0,F0,M0),
      parents(Y0,F0,M0).
```

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X0,Y0):-
      female(X0),
      parents(X0,F0,M0),
      parents(Y0,F0,M0).
```

sister(alice,edward)

↓ X0=alice, Y0=edward

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X0,Y0):-
      female(X0),
      parents(X0,F0,M0),
      parents(Y0,F0,M0).
```

```
7 sister(alice,edward)
          |  X0=alice,
          ↓  Y0=edward
female(alice),
parents(alice,F0,M0),
parents(edward,F0,M0).
```

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,edward)
            |  X0=alice,
            ↓  Y0=edward
```

female(alice),
parents(alice,F0,M0),
parents(edward,F0,M0).

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,edward)
        |  X0=alice,
        ↓  Y0=edward
female(alice),
parents(alice,F0,M0),
parents(edward,F0,M0).
```

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,edward)
        |  X0=alice,
        ↓  Y0=edward
female(alice),
parents(alice,F0,M0),
parents(edward,F0,M0).
        |
        ↓
```

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7  sister(alice,edward)
        |  X0=alice,
        ↓  Y0=edward
3  female(alice),
   parents(alice,F0,M0),
   parents(edward,F0,M0).
        |
        ↓
 parents(alice,F0,M0),
 parents(edward,F0,M0).
```

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,edward)
        |  X0=alice,
        ↓  Y0=edward
3 female(alice),
  parents(alice,F0,M0),
  parents(edward,F0,M0).
        |
        ↓
```

parents(alice,F0,M0),
parents(edward,F0,M0).

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,edward)
        ↓ X0=alice,
          Y0=edward
3 female(alice),
  parents(alice,F0,M0),
  parents(edward,F0,M0).
        ↓
parents(alice,F0,M0),
parents(edward,F0,M0).
```

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,edward)
      |  X0=alice,
      |  Y0=edward
      v
3 female(alice),
  parents(alice,F0,M0),
  parents(edward,F0,M0).
      |
      v
parents(alice,F0,M0),
parents(edward,F0,M0).
      |  F0=victoria,
      |  M0=albert
      v
```

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,edward)
        |  X0=alice,
        |  Y0=edward
        v
3 female(alice),
  parents(alice,F0,M0),
  parents(edward,F0,M0).
        |
        v
6 parents(alice,F0,M0),
  parents(edward,F0,M0).
        |  F0=victoria,
        |  M0=albert
        v
parents(edward,victoria,albert).
```

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,edward)
     |  X0=alice,
     ↓  Y0=edward
3 female(alice),
  parents(alice,F0,M0),
  parents(edward,F0,M0).
     |
     ↓
6 parents(alice,F0,M0),
  parents(edward,F0,M0).
     |  F0=victoria,
     ↓  M0=albert
parents(edward,victoria,albert).
```

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,edward)
      |  X0=alice,
      |  Y0=edward
      v
3 female(alice),
  parents(alice,F0,M0),
  parents(edward,F0,M0).
      |
      v
6 parents(alice,F0,M0),
  parents(edward,F0,M0).
      |  F0=victoria,
      |  M0=albert
      v
5 parents(edward,victoria,albert).
      |
      v
```

Is Alice the sister of Edward?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,edward)
        ↓  X0=alice,
           Y0=edward
3 female(alice),
  parents(alice,F0,M0),
  parents(edward,F0,M0).
        ↓
6 parents(alice,F0,M0),
  parents(edward,F0,M0).
        ↓  F0=victoria,
           M0=albert
5 parents(edward,victoria,albert).
        ↓
        ■
```

Whose sister is Alice?

```
sister(alice,X)
```

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

Whose sister is Alice?

`sister(alice,X)`

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

Whose sister is Alice?

`sister(alice,X)`

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

Whose sister is Alice?

`sister(alice,X)`

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X0,Y0):-
      female(X0),
      parents(X0,F0,M0),
      parents(Y0,F0,M0).
```

Whose sister is Alice?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X0,Y0):-
      female(X0),
      parents(X0,F0,M0),
      parents(Y0,F0,M0).
```

sister(alice,X)

↓ X0=alice, Y0=X

Whose sister is Alice?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
```

(7) `sister(X0,Y0):-`
`female(X0),`
`parents(X0,F0,M0),`
`parents(Y0,F0,M0).`

```
7 sister(alice,X)
      |  X0=alice,
      |  Y0=X
      v
female(alice),
parents(alice,F0,M0),
parents(X,F0,M0).
```

Whose sister is Alice?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
 7 sister(alice,X)
        |  X0=alice,
        ↓  Y0=X
```

female(alice),
`parents(alice,F0,M0),`
`parents(X,F0,M0).`

Whose sister is Alice?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
 7 sister(alice,X)
          |  X0=alice,
          ↓  Y0=X
female(alice),
parents(alice,F0,M0),
parents(X,F0,M0).
```

Whose sister is Alice?

```
(1)  male(albert).
(2)  male(edward).
(3)  female(alice).
(4)  female(victoria).
(5)  parents(edward,
       victoria,
       albert).
(6)  parents(alice,
       victoria,
       albert).
(7)  sister(X,Y):-
       female(X),
       parents(X,F,M),
       parents(Y,F,M).
```

```
  7 sister(alice,X)
        |  X0=alice,
        ↓  Y0=X
female(alice),
parents(alice,F0,M0),
parents(X,F0,M0).
        |
        ↓
```

female(alice) is shown in bold in both (3) and the derivation.

Whose sister is Alice?

```
(1)  male(albert).
(2)  male(edward).
(3)  female(alice).
(4)  female(victoria).
(5)  parents(edward,
       victoria,
       albert).
(6)  parents(alice,
       victoria,
       albert).
(7)  sister(X,Y):-
       female(X),
       parents(X,F,M),
       parents(Y,F,M).
```

```
   7 sister(alice,X)
          |  X0=alice,
          ↓  Y0=X
3 female(alice),
  parents(alice,F0,M0),
  parents(X,F0,M0).
          |
          ↓
 parents(alice,F0,M0),
 parents(X,F0,M0).
```

Whose sister is Alice?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,X)
      |  X0=alice,
      ↓  Y0=X
3 female(alice),
  parents(alice,F0,M0),
  parents(X,F0,M0).
      |
      ↓
```

parents(alice,F0,M0),
parents(X,F0,M0).

Whose sister is Alice?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,X)
        |  X0=alice,
        ↓  Y0=X
3 female(alice),
  parents(alice,F0,M0),
  parents(X,F0,M0).
        |
        ↓
 parents(alice,F0,M0),
 parents(X,F0,M0).
```

Whose sister is Alice?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,X)
    |  X0=alice,
    |  Y0=X
    ↓
3 female(alice),
  parents(alice,F0,M0),
  parents(X,F0,M0).
    |
    ↓
parents(alice,F0,M0),
parents(X,F0,M0).
    |  F0=victoria
    |  M0=albert
    ↓
```

Whose sister is Alice?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
        7 sister(alice,X)
               |  X0=alice,
               ↓  Y0=X
     3 female(alice),
       parents(alice,F0,M0),
       parents(X,F0,M0).
               |
               ↓
     6 parents(alice,F0,M0),
       parents(X,M0,F0).
               |  F0=victoria
               ↓  M0=albert
parents(X,victoria,albert).
```

Whose sister is Alice?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
 7 sister(alice,X)
        |  X0=alice,
        ↓  Y0=X
 3 female(alice),
   parents(alice,F0,M0),
   parents(X,F0,M0).
        |
        ↓
 6 parents(alice,F0,M0),
   parents(X,M0,F0).
        |  F0=victoria
        ↓  M0=albert
parents(X,victoria,albert).
```

Whose sister is Alice?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
7 sister(alice,X)
        |  X0=alice,
        ↓  Y0=X
3 female(alice),
  parents(alice,F0,M0),
  parents(X,F0,M0).
        |
        ↓
6 parents(alice,F0,M0),
  parents(X,M0,F0).
        |  F0=victoria
        ↓  M0=albert
5 parents(X,victoria,albert).
        |  X=edward
        ↓
```

Whose sister is Alice?

```
(1) male(albert).
(2) male(edward).
(3) female(alice).
(4) female(victoria).
(5) parents(edward,
      victoria,
      albert).
(6) parents(alice,
      victoria,
      albert).
(7) sister(X,Y):-
      female(X),
      parents(X,F,M),
      parents(Y,F,M).
```

```
          7 sister(alice,X)
               |  X0=alice,
               ↓  Y0=X
   3 female(alice),
     parents(alice,F0,M0),
     parents(X,F0,M0).
               |
               ↓
   6 parents(alice,F0,M0),
     parents(X,M0,F0).
               |  F0=victoria
               ↓  M0=albert
5 parents(X,victoria,albert).
               |  X=edward
               ↓
               ■
```

Answer: X = edward.

Exercise: thieves

Predicates that are being used:

> thief(X): "X is thief."
> likes(X, Y): " X likes Y."
> may_steal(X, Y): "X may steal Y."

Ştim că:

1. X may steal Y if X is thief and X likes Y.
2. John is thief.
3. Mary likes food
4. Mary likes wine.
5. John likes X if X likes wine.

Query: What may John steal?

- Knowledge base:

```
thief(john).
likes(mary, food).
likes(mary, wine).
likes(john, X) :- likes(X, wine).
may_steal (X, Y) :-
        thief(X), likes(X, Y).
```

- Query:

```
?- may_steal(john,X).
```

Sometimes, Prolog does not behave as expected.

Example

```
\% father(X,Y) means that the father of X is Y
father(maria,george).
father(ion,george).
father(elena,eric).
?- father(_,X).
   X=george;
   X=george;
   X=eric;
```

1. PROLOG returns twice the answer (`X=george`) because there are 2 facts which confirm that `X` is the father of somebody.
2. We wish ti have a way to avoid the generation of repeated answers.

Example

```
nat(0).
nat(X):-
  nat(Y),
  X is Y+1.

?-nat(X).
   X=0 ;
   X=1 ;
   X=2 ;
   X=3;
   ...
```

This is the expected behavior of PROLOG!

Finding answers by backtracking

Example

```
member(X,[X|_]).
member(X,[_|T]):-member(X,T).

?-member(a,[b,a,d,a,c]).
   true ;
   true ;
   false .
```

- The backtracking process confirms the answers as many times as `a` occurs in the list.
- It is sufficient to get one confirmation.

More about PROLOG computations

Finding answers by backtracking

```
member(X,[X|_]).
member(X,[_|T]):-member(X,T).
```

```
?-member(a,[b,a,d,a,c]).
```

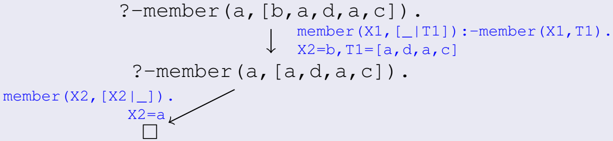
Finding answers by backtracking

```
member(X,[X|_]).
member(X,[_|T]):-member(X,T).
```

```
?-member(a,[b,a,d,a,c]).
        ↓   member(X1,[_|T1]):-member(X1,T1).
            X2=b,T1=[a,d,a,c]
 ?-member(a,[a,d,a,c]).
```

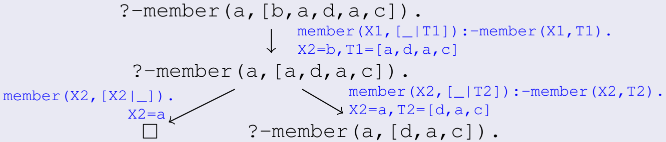
Finding answers by backtracking

```
member(X,[X|_]).
member(X,[_|T]):-member(X,T).
```



Finding answers by backtracking

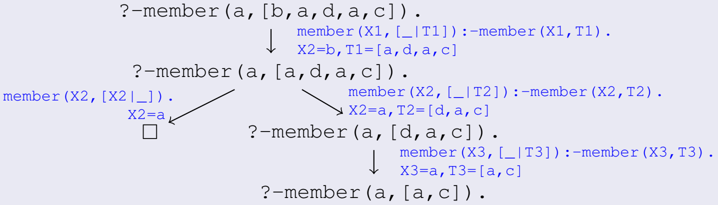
```
member(X,[X|_]).
member(X,[_|T]):-member(X,T).
```



More about PROLOG computations

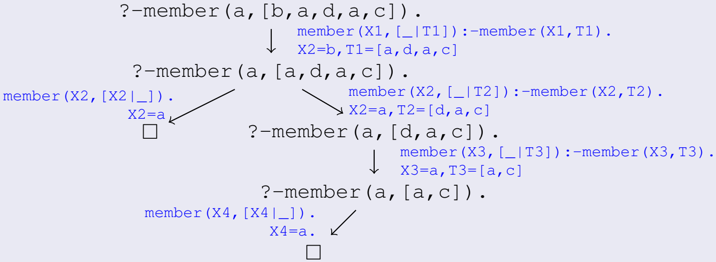
Finding answers by backtracking

```
member(X,[X|_]).
member(X,[_|T]):-member(X,T).
```



Finding answers by backtracking

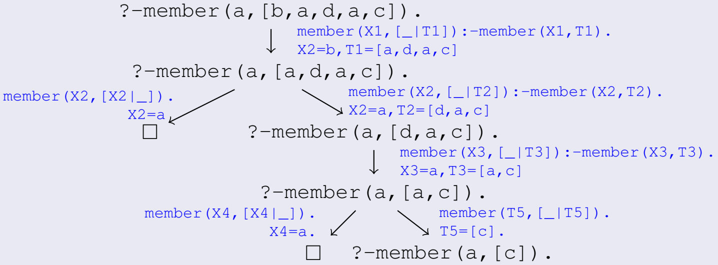
```
member(X,[X|_]).
member(X,[_|T]):-member(X,T).
```



More about PROLOG computations

Finding answers by backtracking

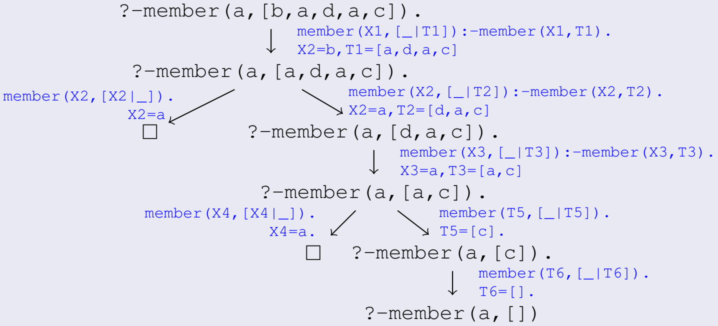
```
member(X,[X|_]).
member(X,[_|T]):-member(X,T).
```



More about PROLOG computations

Finding answers by backtracking

```
member(X,[X|_]).
member(X,[_|T]):-member(X,T).
```



Conclusion

We indicated the main principles of Logic Programming.

- facts, conjunctions of facts, logical variables
- rules
- examples which illustrate
  - How to program in PROLOG
  - How does PROLOG find answers to the queries of the user

References, exercises

- You should become familiar with logic programming using SWI Prolog.
- All details about SWI Prolog can be found at `http://www.swi-prolog.org`.
  Install SWI-Prolog on your laptop, or use the version available in lab rooms, and try the examples described in this lecture.