Introduction to Logic Programming and Prolog

Module of Logics and Artificial Intelligence course

Logic Programming

The Idea

Instead of writing down an algorithm to solve a specific problem, we can **describe** the environment and the problem and let **the program find** the solution, if exists, for us.

Logic Programming

Key features:

- Logic Programming is a **Programming Paradigm** based on **formal logic**;
- Logic is considered formal when its sentences are represented in the formal grammar, syntax and semantics of a logical language, suitable for **formal inference**;
- In a logic program, data and goals are expressed through clauses, in a declarative form.

Differences between Procedural and Declarative Languages

- In **Procedural languages** a sequence of instructions is defined in order to implement an algorithm. A typical procedural language is C99.
  The focus is on how the computation has to be performed.

- **Declarative languages** programs consist in a formal description of the wanted information. A typical declarative language is SQL.
  The focus is on what computation has to be performed.

A logic program consists in a sequence of facts (the true facts about the world), and rules, that let the program to infer implicit knowledge from the facts.

- The **problem** to solve (the **goal**) is **represented** in a **logical language**
- **The solving** of the problem is **realized** by **deductive reasoning**

A classical example

All men are mortal (a **rule**)
Socrates is a man (a **fact**)
It can be **inferred** that Socrates is mortal

Defining relations by facts

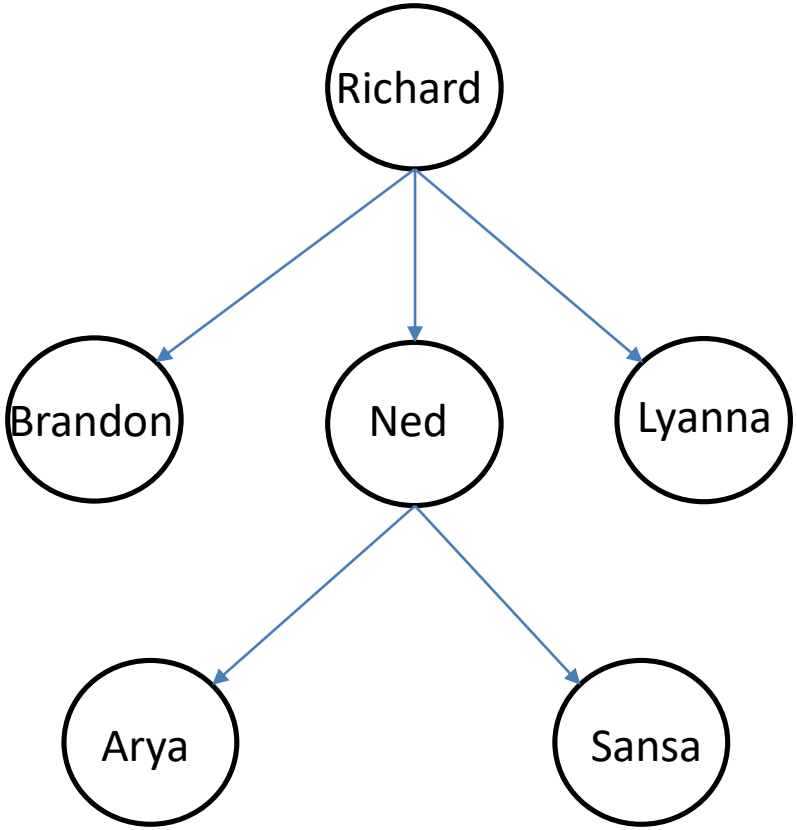
Prolog is specially well suited for solving problems that involve objects and relations between objects.

Let's see an example:
The fact that **Ned is a parent of Arya** can be written in Prolog as:

parent(ned,arya).

parent: name of the relation
ned, **arya**: arguments of the relation

A family tree example:




```
parent(richard,ned).
parent(richard,brandon).
parent(richard,lyanna).
parent(ned,arya).
parent(ned,sansa).
```

A closer view:

```
parent(richard,ned).
parent(richard,brandon).
parent(richard,lyanna).
parent(ned,arya).
parent(ned,sansa).
```

- The program consists of five clauses.

Each clause declares one fact about the parent relation, it is **a particular instance of the relation**.

- Order matters.

The **order of the clauses is important**.

- Each Prolog **program represents the knowledge base**

- Closed World Assumption

It is assumed that **whatever is neither explicitly nor implicitly expressed is false**

Query

A Prolog program can be run executing a query, the goal, against its knowledge base.

The query must be written in the same formal logic.

The program will answer true if it finds the clause as an asserted fact in the program

?- parent(richard,ned).
true

?- parent(arya,ned).
false

Query

In query clauses variables can be used:

Who is Ned's father?
?- parent(X,ned).
X = richard

The variables begins with a capital letter
The strings with an initial lower case letter are constants.

Who is Ned's child?
?- parent(ned,X).
X = arya

The program returned the first fact that makes the query true. If another solution is requested, it is possible to type a semicolon(;):
X = sansa

Query

It can be asked a broader question:
Find X and Y such that X is a parent of Y

?- parent(X,Y).

X= richard
Y= ned;
X= richard
Y= ned;
X= richard
Y= ned;
X= ned
Y= arya;
X= ned
Y= sansa.

Query

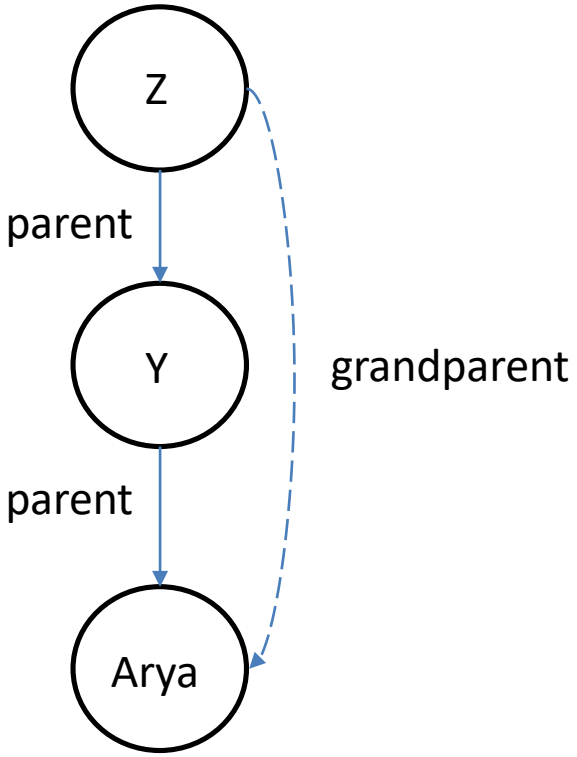
More complex questions can be posed: Who is the grandfather of Arya?
This query can be expressed by two simpler queries:

1) Find Y such that Y is parent of Arya.
2) Find Z such that Z is parent of Y

?- parent(Y,arya),parent(Z,Y).

The answer will be:

Z = richard
Y = ned.



Query

Other examples:

Who are Richard's grandchildren?

Or

Do Arya and Bran have a common parent?

Query

Other examples:

Who are Richard's grandchildren?

?- parent(richard,Y),parent(Y,X).

Or

Do Arya and Bran have a common parent?

?- parent(Z,arya),parent(Z,bran).

Recap

- A **relation** such as parent relation can be **easily defined** in Prolog
- A **prolog program consists of clauses**, each clause terminates with a full stop.
- The arguments of a relation can be:
  - concrete objects or **constants** (**atoms**)
  - general objects as X,Y,Z (**variables**)
  - other **relations**
- It is easy to query Prolog system about relations
- **Queries** to Prolog consists of one or more **goals**
- An **answer** can be positive or negative. If the answer is positive we say that the **goal** was **satisfiable** and the goal succeded. Otherwise the goal was **unsatisfiable** and the goal failed.
- If semicolon is typed, Prolog will find for further satisfying answers

Defining relations by rules

Consider this logical statement:

For all X and Y,
X is the father of Y
if X is a parent of Y, and X is a male.

This statement can be written in Prolog as:

father(X,Y) :- parent(X,Y),male(X).

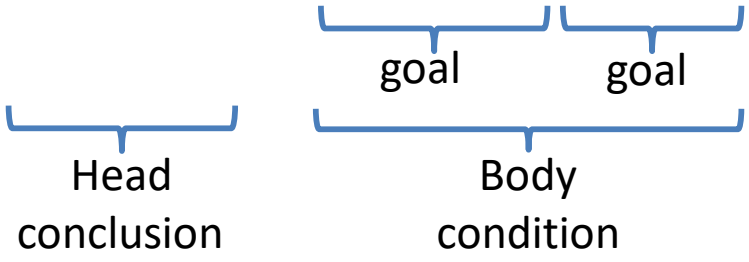
This kind of clauses are called **rules**.

The rules

A fact like parent(ned,arya) or male(ned) **is always and unconditionally true**

In a **rule**, instead, **a fact is true if some condition is satisfied**.

father(X,Y) :- parent(X,Y),male(X).



If the **goals** of the **body** (the **condition**) are true then the **head** (the **conclusion**) is a logical consequence of this and is true.

An example

?- father(ned,arya).
There are no facts about father in the program so applying the rule about father is needed.

The rule can be applied to any X and Y, in this case X has to be substituted with ned and Y with arya.
X = ned
Y = arya

The rule with instantiated variables becomes:
father(ned,arya):-parent(ned,arya),male(ned).
with
parent(ned,arya),male(ned) condition part
father(ned,arya) conclusion part

An example

The goal
father(ned,arya)

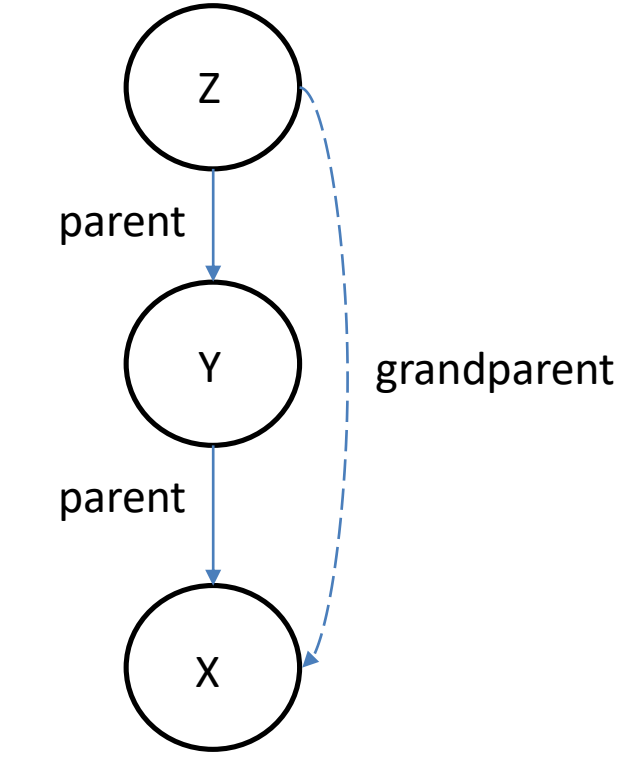
becomes
parent(ned,arya),male(ned)

?- parent(ned,arya)
True
and
?-male(ned)
True

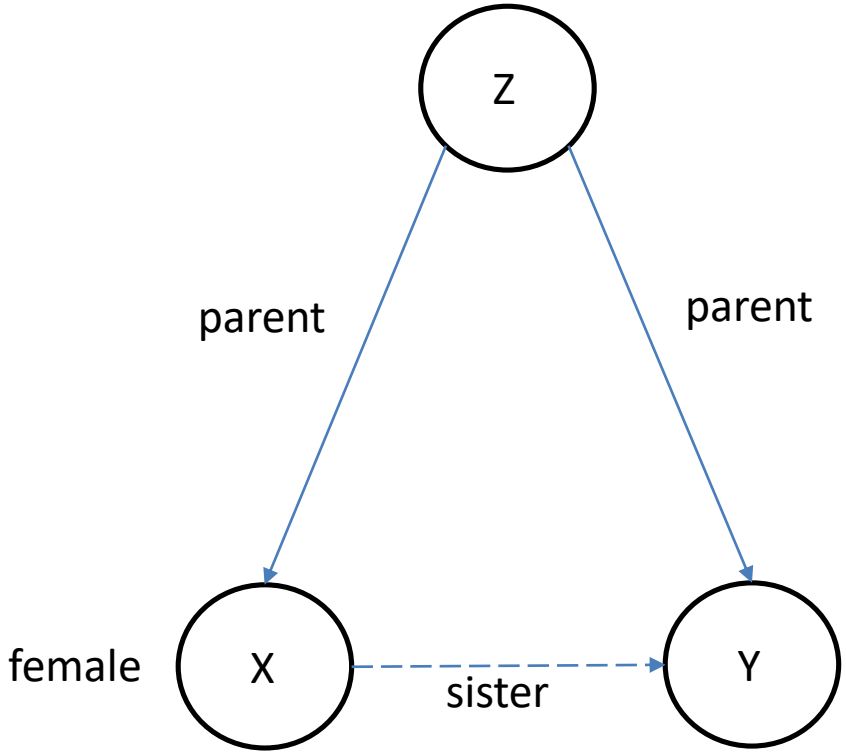
These goals can be easily found as facts in the program, hence Prolog can answer **true**

An example

Similarly we can express the **grandparent relation** and the **sibling relation**.



grandparent relation



sibling relation

Recap

- **Facts** declare things that **are unconditionally true**
- **Rules** declare things that **are true** depending **on a given condition**
- A Prolog **clause consists** of **head** and **body**
- The **body can be a composition of** comma separated **goals** (a conjuction of goals)
- This kind of clauses are called **Horn clauses**

What are Horn clauses?

A **Horn clause** is a clause with **exactly one positive literal**

A **clause** is a **disjunction of literals**

A **literal** is an **atomic formula** or its negation

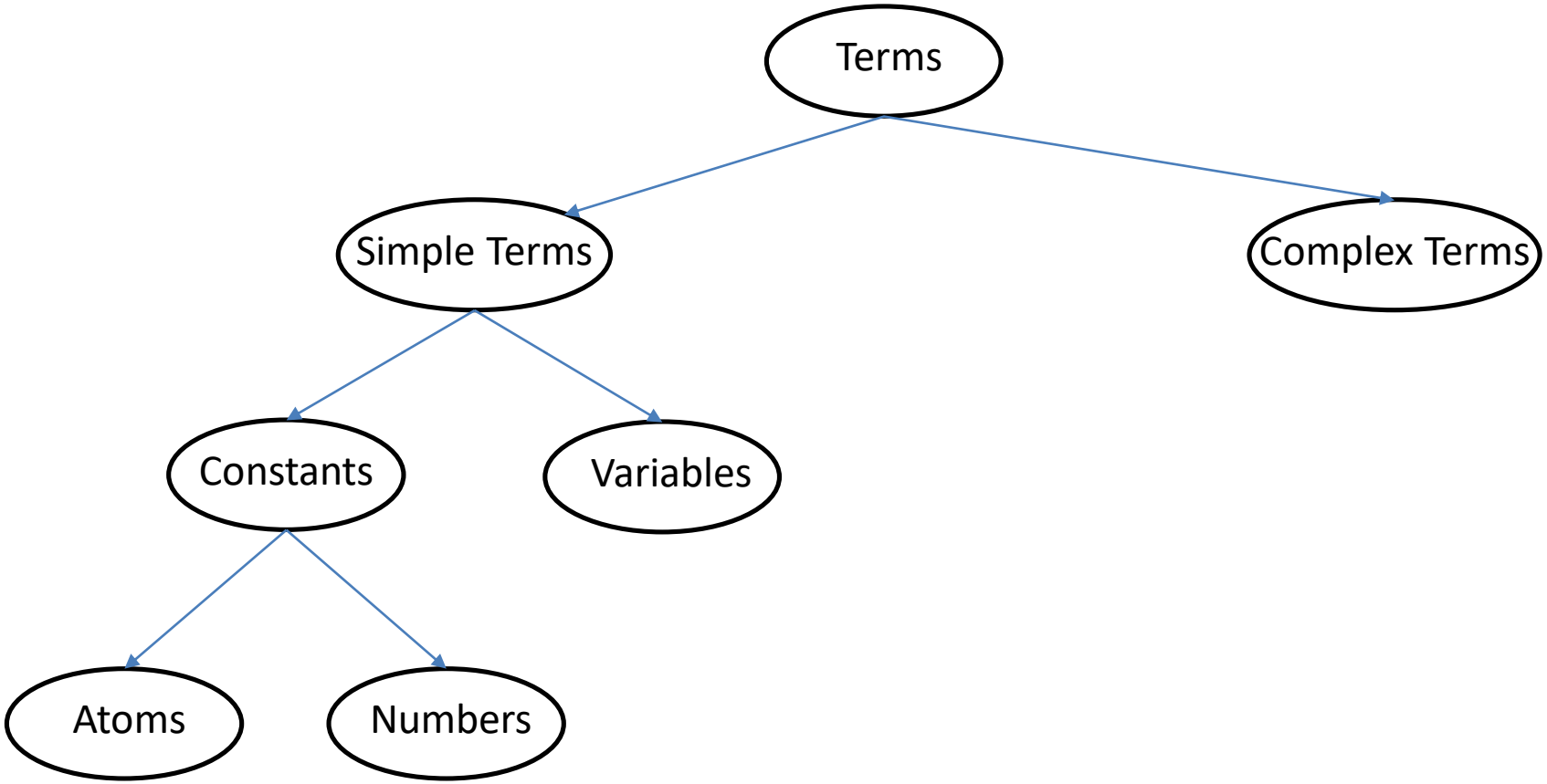
A **Horn formula** is a **conjunctive normal form formula** whose **clauses are all Horn**

Prolog: c:- a, b.
a.
b.

Horn formula: $[c \vee \neg a \vee \neg b] \wedge a \wedge b$

Prolog Syntax

Facts, rules and queries are **built on Prolog terms**.



Atoms

- A sequence of characters of upper-case letters, lower-case letters, digits, or underscore, starting with a lowercase letter
  Examples: ned, arya, parent

- An arbitrary sequence of characters enclosed in single quotes
  Examples: ‘X', Y', '@$%‘

- A sequence of special characters
  Examples: : , ; . :-

Numbers

- **Integers**: 12, -34, 22342
- **Floats**: 34573.3234, 0.3435

Variables

A sequence of characters of uppercase letters, lower-case letters, digits, or underscore, starting with either an uppercase letter or an underscore

Examples:
X, Y, Variable, Vincent, _tag

Complex Terms

Atoms, numbers and variables are building blocks for complex terms

Complex terms are built out of a **functor** directly followed by a sequence of arguments

- Arguments are put in **round brackets**, separated by commas
- The functor must be an atom

Examples:

- male(ned)
- parent(ned, arya)
- granchild(arya, Z)
- father(X,father(father(father(arya))))

Arity

The number of arguments a complex term has is called its **arity**

Examples:

female(sansa)	has arity 1
sibling(arya,bran)	has arity 2
father(X,father(ned,Z))	has arity 2

Several predicates can be defined with the same functor but **with different arity**, and Prolog would treat this as **different predicates**

Arity of a predicate is usually indicated with the suffix "**/**" followed by a number to indicate the arity

female/1
sibling/2
father/2