Brief Introduction to Prolog

Joana Côrte-Real
jcr@dcc.fc.up.pt

CRACS & INESC TEC
Faculty of Sciences
University of Porto

University of Aizu
5th December 2014

Overview

Section 1

Introduction

Programming paradigms

Imperative translation from machine language to user commands

Object object-oriented, versatile and recent

Declarative detachment between program's goal and execution details

- **Functional** concerned with recursion, pattern matching, ...
- **Logic** focus on automatically reasoning about knowledge in the program

Introduction to Prolog

- Prolog represents **Horn Clauses** - which are a subset of **First Order Logics** - where each clause can have at most one positive literal in the **head**.

$$\text{head} :- \text{body}.$$

- A Prolog program is a set of **facts** and/or **rules** defining relations between entities.
- **Facts** represent relations which are assumed to be true (axioms)
- **Rules** can be true or false, depending on other relations in the program
- A Prolog program is resolved by computing the consequents of rules.

Prolog syntax 1

Characters Letters and numbers

Terms can be

- **variables** **Person**, **_Father**
- **constants** **mary**, **'The Family'**
- **compound** **family('mary&tom', date(25, feb, 1954))**

Symbols `:-` `;` `,` `.`

Comments can be used for

- **line** %Comment
- **text** /* Also a comment */

Prolog syntax 2

Operators + - * /

Unification = finds a set of variable substitutions that make two terms exacty the same.

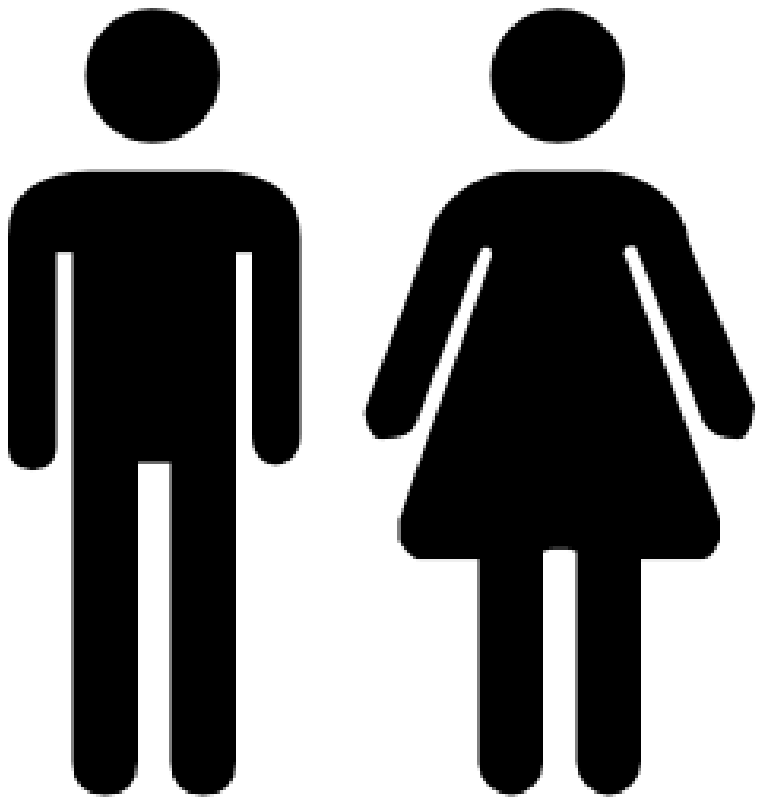
Arithmetic Prolog does not evaluate mathematical expressions unless the operator **is** is used

numbers **is** > < =\=

terms == \==

Family - facts

```
%Facts T1
mother_of(mary,john).
mother_of(mary,anne).
father_of(tom,anne).
father_of(tom,john).

%Query T1
?-mother_of(mary,Person).
```



Observations

- There are two predicates in this example: **mother_of/2** and **father_of/2**.
- **pred/N** represents predicate **pred(arg1, arg2, ..., argN)**, with *arity* of N arguments.
- These predicates have no body and so they are called *facts*.
- Facts can be seen as a multi-relational database in Prolog.

Section 2

Prolog data structures

Data structures: List

A few examples:

- []: empty list.
- [the,men, [like,to,fish]]
- [a,V1,b, [X,Y]]

List is composed of: **[Head|Tail]**
Head: first element of the list (can be of any type).
Tail: remaining elements as a list.

List	Head	Tail
[a,b,c]	a	[b,c]
[a]	a	[]
[]	no head	no tail

Equality of Lists

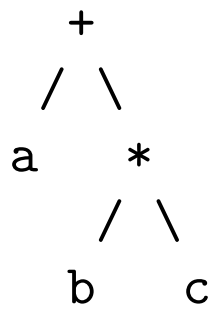
- Unifications can happen inside lists
- The empty list does not have head or tail
- This is used as a stop criterion in list recursion

Examples:

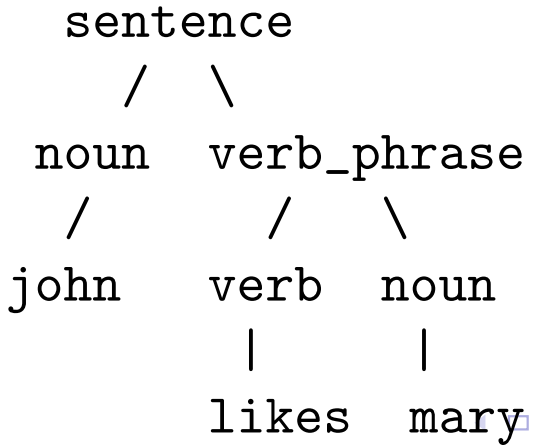
`[X,Y,Z]`	=	`[john,likes,fish]`	`X=john, Y=likes, Z=fish`
`[cat]`	=	`[X|Y]`	`X=cat, Y=[]`
`[]`	=	`[X|Y]`	will always fail

Data structures: Tree

Tree below can be represented as either **a+b*c** or **+(a,*(b,c))**



Trees can be used to represent concept such as:
sentence(noun(john),verb_phrase(verb(likes),noun(mary)))



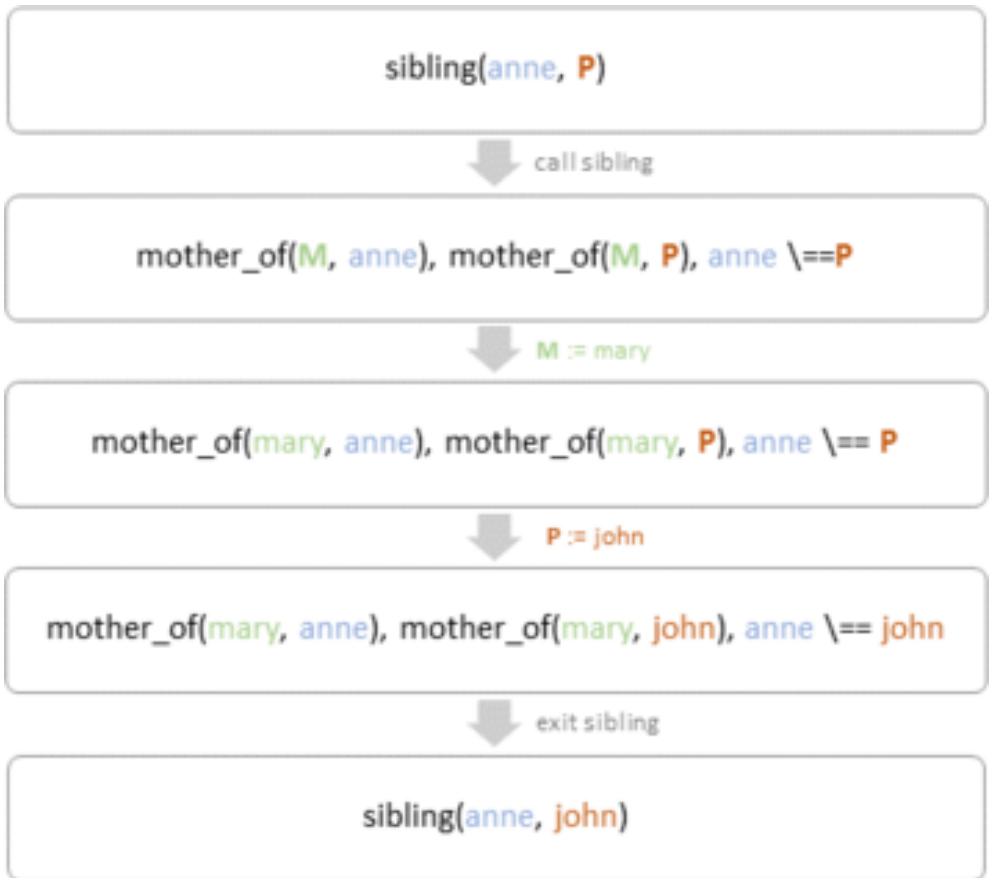
Family - rules

```
%Facts T2
father_of(tom, bill).
mother_of(mary, jane).

%Rule T2
sibling(Person1, Person2):-
    mother_of(Mother, Person1),
    mother_of(Mother, Person2),
    Person1\==Person2.
sibling(Person1, Person2):-
    father_of(Father, Person1),
    father_of(Father, Person2),
    Person1\==Person2.

%Query T2
?-sibling(anne, Person).
```

Observations

- **findall/3** predicate gathers all possible solutions for the query in argument 2 in a list.
- Procedural interpretation: to solve **sibling/2**, predicates **mother_of/2** and **father_of/2** must be evaluated first.
- The resolution mechanism (SLD resolution) builds an execution tree.

Execution tree for **sibling**



Section 3

Backtracking

Obtaining multiple solutions in Prolog

- **Backtracking** in Prolog can be used to obtain multiple solutions for a goal (predicate fails vs user-induced failure).
- When Prolog is computing a goal, every choice it makes is stored as a **choice point**.
- If a given goal fails, it can be backtracked (choices undone) until the previous choice point is restored.
- From there, Prolog starts looking for a new alternative for solving the goal.

Family - recursive

```
%Facts T3
father_of(roger,tom).
father_of(greg,roger).

%Rule T3
male_ancestor(Person1,Person2,1):-
   father_of(Person1,Person2).
male_ancestor(Person1,Person2,Level):-
   father_of(Person1,NewPerson),
   male_ancestor(NewPerson,Person2,
        NewLevel),
   Level is NewLevel+1.

%Query T3
?-male_ancestor(P,anne,2).
```

Observations

- Declarative interpretation of query: are **greg** and **anne** related through the male line?
- **male_ancestor/3** can solve both the level of the relation and find a person who is related to someone else.
- Note that recursion must be made with new variables every time: **NewLevel**, **NewPerson**.

Execution tree for **male_ancestor**

Section 4

Conclusion

Conclusions

- Prolog allows for an almost direct match between a FOL algorithm and its execution syntax.
- Because Prolog is based on FOL, if the facts are true, then the program will produce true results.
- Prolog's declarative nature is particularly suited to build and consult multi-relational data.

There are many Prolog systems in the literature with different features such as Yap Prolog[1], SWI-Prolog[2], SICStus Prolog[3], etc.

[1] www.dcc.fc.up.pt/ vsc/Yap/documentation.html
[2] www.swi-prolog.org
[3] sicstus.sics.se

Thank you

Joana Côrte-Real