Utrecht University

Introduction to Prolog

An Overview

Alvaro H. C. Correia

23rd of May 2019

Utrecht University

Course Info

Material

- **Learn Prolog Now!**

  Whole book, excluding chapters 7 and 8

  Free online verstion at `http://www.learnprolognow.org/`
- **Bratko, Prolog - Programming for Artificial Intelligence**

  Part 1 and Part 2:ch.14
- **Shapiro, The Art of Prolog**

Prolog Implementation

We will use the SWI-Prolog implementation during the course.

`http://www.swi-prolog.org/`

Overview of Prolog

Declarative vs Imperative Programming

- **Imperative**

  How to solve the problem

  A program is composed of a set of ordered instructions.

- **Declarative**

  What is the problem

  A program is a set of *facts* and *rules*.

Declarative vs Imperative Programming (2)

- **Imperative**
  - Most programming languages.
    - Java, C/C++, Python, Lisp, Ruby...
- **Declarative**
  - Logic programming languages.
    - Prolog the most important and well established.
  - Few domain-specific languages
    - SQL, HTML.

So what is Prolog?

Prolog - **PRO**grammation **LOG**ique

Alain Colmerauer and *Philippe Roussel* - Marseilles, 1972.

- Designed for Natural Language Processing.
- Operates with both numbers and **symbols**.
- Popular for classical AI and **knowledge-based systems**.

Facts, Rules and Queries

- A **Fact** states something that is unconditionally (always) true.

  ```
  fun(prolog).
  ```

- A **Rule** states something that is true if a given condition holds.

  ```
  fun(X) :- loves(Y, X).
  ```

- A **Query** verifies whether a fact is true.

  ```
  ?- loves(alvaro, prolog).
  ```

  A **Query** also finds an object for which a fact is true.

  ```
  ?- loves(X, prolog).
  ```

Facts

Atoms and Predicates

`parent(vader, luke).`

- `parent` is a **functor**, or a **predicate**.
- `vader` and `luke` are **atoms**.
- The **arity** is the number of atoms in a predicate.
- `parent/2`

We can read this fact as ”`vader` is a `parent` of `luke`.”

Some facts

```
raining.

nice.

parent(vader).

plays(federer, tennis).

beats(brazil, argentina).

book(
    harryPotter,
    author(joanne, rowling),
    bloomsbury,
    1997
).
```

A First Program

```
pizza(marg). % margherita
pizza(mari). % marinara
pizza(napo). % napoletana
contains(marg, moz). % mozzarella
contains(marg, bas). % basil
contains(mari, gar). % garlic
contains(mari, ore). % oregano
contains(napo, moz).
contains(napo, ore).
contains(napo, anc). % anchovies
```

Queries

Making Queries - True or False

Asking Prolog whether a fact is true or false

We interact with Prolog through queries.

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

```
?- pizza(marg).
```

Making Queries - True or False

Asking Prolog whether a fact is true or false

Prolog answers `true` if it finds that fact in the program.

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

```
?- pizza(marg).
true.
```

Making Queries - True or False

Asking Prolog whether a fact is true or false

If it does not follow from the program, it is false (closed world assumption).

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

```
?- pizza(marg).
true.
?- contains(marg, anc).
false.
```

Making Queries - True or False

Asking Prolog whether a fact is true or false

We can also ask about atoms that do not appear in the program.

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

```
?- pizza(marg).
true.
?- contains(marg, anc).
false.
?- pizza(pepperoni).
false.
```

Making Queries - True or False

Asking Prolog whether a fact is true or false

We can also ask about predicates that do not appear in the program.

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

```
?- pizza(marg).
true.
?- contains(marg, anc).
false.
?- pizza(pepperoni).
false.
?- tasty(napo).
false.
```

Making Queries - Searching for an Atom

Suppose that we want a pizza with anchovies

We could pose a series of questions.

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

```
?- contains(marg, anc)
false.
?- contains(mari, anc).
false.
?- contains(napo, anc).
true.
```

Variables and Existential Queries

A **variable** in Prolog is an unspecified individual or object.

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

Making Queries - Using Variables

Variables and Existential Queries

Does there exist an X such that X contains anc?

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

```
?- contains(X, anc).
X = napo.
```

Variables and Existential Queries

Does there exist an X such that marg contains X?

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

```
?- contains(X, anc).
X = napo.
?- contains(marg, X).
X = moz;
X = bas.
```

Variables and Existential Queries

Does there exist a pair X and Y such that X contains Y?

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

```
?- contains(X, anc).
X = napo.
?- contains(marg, X).
X = moz;
X = bas.
?- contains(X, Y).
X = marg;
Y = moz.
```

A bit of syntax

Naming atoms and variables

- A **variable** always start with an upper-case letter or an underscore.

  `X, Y, Output, Answer, _x, _y`

- An **atom** starts with a lower-case letter.

  `utrecht, netherlands, bear, x, y`

When a variable is set to an atom, e.g. `X = napo`, we say that the variable is **bound** or **instantiated** to that atom.

When a variable is not set, we say it is a **free** variable.

A bit of syntax

Naming atoms and variables

- A **variable** always start with an upper-case letter or an underscore.

  `X, Y, Output, Answer, _x, _y`

- An **atom** starts with a lower-case letter.

  `utrecht, netherlands, bear, x, y`

When a variable is set to an atom, e.g. `X = napo`, we say that the variable is **bound** or **instantiated** to that atom.

When a variable is not set, we say it is a **free** variable.

We can only instantiate a variable once!

Matching

Given two terms, we say that they match (or unify) if

- They are identical;
- The variables in both terms can be instantiated to objects in such a way that after the substitution of variables by these objects the terms become identical.

Ivan Bratko. *Prolog Programming for Artificial Intelligence*. 3rd ed. Harlow, England: Pearson Addison-Wesley, 2000. ISBN: 978-0-201-40375-6.

Matching

Examples of both cases

- `pizza(marg)` *unifies with itself (and hence is true).*
- `pizza(X)` *unifies with* `pizza(marg)`, `pizza(mari)`, `pizza(napo)` *with* `X=marg`, `X=mari`, `X=napo`, *respectively*.

How Prolog searches for an answer

Suppose we want to find a pizza with oregano and anchovies

We can make a series of queries.

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

```
?- contains(X, ore).
X = mari.
```

How Prolog searches for an answer

Suppose we want to find a pizza with oregano and anchovies

We can make a series of queries.

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

```
?- contains(X, ore).
X = mari.
?- contains(mari, anc).
false.
```

How Prolog searches for an answer

Suppose we want to find a pizza with oregano and anchovies

Our first attempt failed. Go back to the first query.

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

```
?- contains(X, ore).
X = mari ;
X = napo.
```

How Prolog searches for an answer

Suppose we want to find a pizza with oregano and anchovies

Our first attempt failed. Go back to the first query.

```
pizza(marg).
pizza(mari).
pizza(napo).
contains(marg, moz).
contains(marg, bas).
contains(mari, gar).
contains(mari, ore).
contains(napo, moz).
contains(napo, ore).
contains(napo, anc).
```

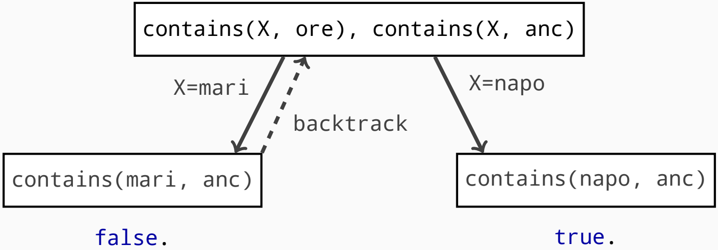
```
?- contains(X, ore).
X = mari ;
X = napo.
?- contains(napo, anc).
true.
```

How have we solved this problem?

1. Define our problem as a list of queries. We call each of those a **goal**.
2. Try to prove each goal in order, from left to right.
3. If a goal has a match in the program, proceed to the next goal, while keeping variable instantiations.
   3.1 If the goal has a free variable, we bind that variable to an atom that produces a match.
   3.2 If there is other possible bindings, we mark the goal as a choicepoint.
4. If a goal has no match in the program,
   4.1 return to the last choicepoint - **backtrack**.
   4.2 if all choicepoints have been checked, return false.
5. If we found matches for all goals, return true and all the variable instantiations.

Search Tree

Nodes are goals. Edges are variable instantiations.



Conjunction - Disjunction

Conjunction

Logical `and`: both goals must be true.

Denoted by a comma.

```
contains(X, ore), contains(X, anc).
```

Disjunction

Logical `or`: at least one of the goals must be true.

Denoted by a semicolon.

```
contains(X, ore); contains(X, anc).
```

Conjunction - Disjunction

Conjunction

Logical `and`: both goals must be true.

Denoted by a comma.

```
contains(X, ore), contains(X, anc).
```

Disjunction

Logical `or`: at least one of the goals must be true.

Denoted by a semicolon.

```
contains(X, ore); contains(X, anc).
```

If a variable appears in more than one goal in a conjunctive query, it must be instantiated to the same atom in all goals.

Universal Facts

Defining universal truth

Suppose we want to include in our program that every pizza contains tomato sauce.

Universal Facts

Defining universal truth

We could write a new fact for every pizza in our program.

```
contains(marg, tomato_sauce).
contains(mari, tomato_sauce).
contains(napo, tomato_sauce).
```

Universal Facts

Defining universal truth

Variables allow us to define a *fact* that holds for every *atom*.

```
contains(marg, tomato_sauce).
contains(mari, tomato_sauce).
contains(napo, tomato_sauce).
contains(X, tomato_sauce).
```

Universal Facts

Defining universal truth

Variables allow us to define a *fact* that holds for every *atom*.

```
contains(marg, tomato_sauce).
contains(mari, tomato_sauce).
contains(napo, tomato_sauce).
contains(X, tomato_sauce).
```

Variables can represent anything!

```
?- contains(cat, tomato_sauce).
true.
```

Anonymous Variable

When to use the anonymous variable _

In our last example X was a singleton variable.

```
contains(X, tomato_sauce).
```

Warning: Singleton variables [X].

Anonymous Variable

When to use the anonymous variable _

Variables convey information from one place to the other.

> *"Using a variable only once is nonsense."*

```
contains(X, tomato_sauce).
```

Warning: Singleton variables [X].

Anonymous Variable

When to use the anonymous variable _

When a variable appears only once or we do not care how it is bound, we use an anonymous variable.

```
contains(X, tomato_sauce).
```

Warning: Singleton variables [X].

```
contains(_, tomato_sauce).
```

Anonymous Variable

When to use the anonymous variable _

When we use an anonymous variable, Prolog does not show us its binding.

```
?- pizza(X). % Find me a pizza X
X = marg.
?- pizza(_). % Is there any pizza at all?
true.
```

Anonymous Variable

When to use the anonymous variable _

Every occurrence of the name of a variable stands for the same variable.

Every occurrence of an underscore stands for a different variable.

```
?- contains(X, moz), contains(X, gar).
false.
?- contains(_, moz), contains(_, gar).
true.
```

Rules

Rules

General structure

`head :- body.`

”head is true if body is true.”

Rules

General structure

`head :- body.`

"head is true if body is true."

```
contains(X, tomato_sauce) :- pizza(X).
```

Rules

General structure

```
head :- body.
```

"head is true if body is true."

```
contains(X, tomato_sauce) :- pizza(X).
```

If `X` *is a* `pizza`, `X` `contains` `tomato_sauce`.

```
?- contains(cat, tomato_sauce).
false.
```

Rules

General structure

`head :- body.`

”head is true if body is true.”

```
contains(X, tomato_sauce) :- pizza(X).
contains(X, tomato_sauce) :- pasta(X).
```

Rules

General structure

`head :- body.`

"head is true if body is true."

```
contains(X, tomato_sauce) :- pizza(X).
contains(X, tomato_sauce) :- pasta(X).
```

If `X` *is a* `pizza` ***or*** *a pasta,* `X` `contains` `tomato_sauce`.

We can have many rules for a single predicate.

The set of rules that define a predicate is called a **procedure**.

Rules

General structure

```
head :- goal_1, goal_2, ..., goal_n.
```

"head is true if each one of the goals is true."

```
cheese_pizza(X) :- pizza(X), contains(X, moz).
```

Rules

General structure

```
head :- goal_1, goal_2, ..., goal_n.
```

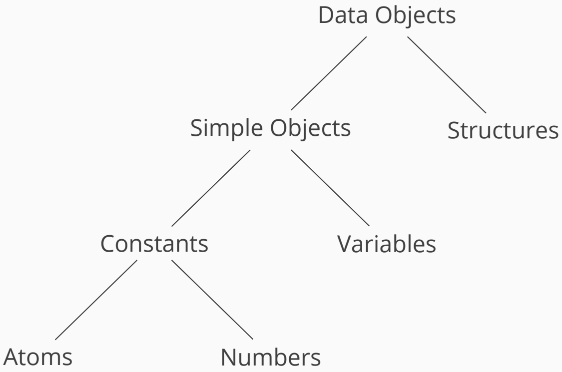
"head is true if each one of the goals is true."

```
cheese_pizza(X) :- pizza(X), contains(X, moz).
```

If `X` *is a pizza* ***and*** `X` `contains` `moz`*,* `X` *is a* `cheese_pizza`.

Data Structures

Data Structures



[1]

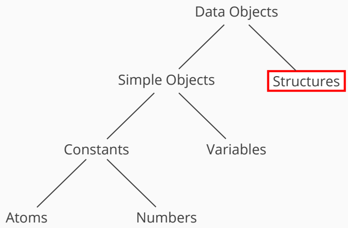
---

[1] Ivan Bratko. *Prolog Programming for Artificial Intelligence*. 3rd ed. Harlow, England: Pearson Addison-Wesley, 2000. ISBN: 978-0-201-40375-6.

Structures

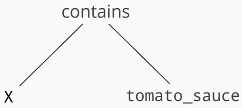
Structures (or complex terms) are composed of a functor followed by a series of components.
The components might be simple objects or structures themselves.

```
date(23, may, 2019).
foo(bar(X), bah(Y,Z)).
contains(X, tomato_sauce).
```

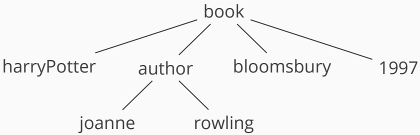


We can also represent structures as trees.

```
contains(X, tomato_sauce).
```



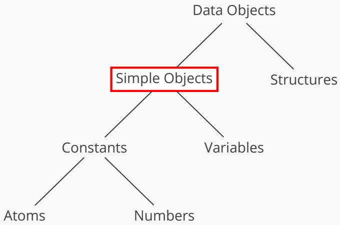

```
book(
    harryPotter,
    author(joanne, rowling),
    bloomsbury,
    1997
).
```



Simple Objects

Simple objects are formed by a single component, which can be either a constant or a variable.

```
X.
tomato.
42.
```

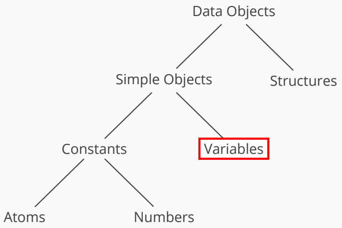


Variables

Variables represent any but always the same element.
The anonymous variable represents any but always a different element.

Roles of variables:

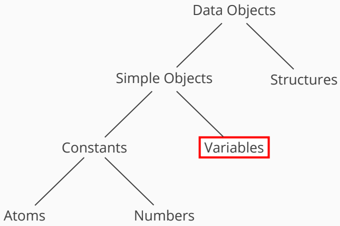
- Represent unknown elements;
- Place-holder;
- Coreference constraint.



Variables

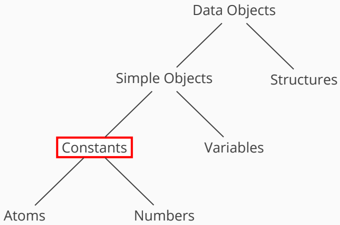
Variables are instantiated only once.

An assignment is only annulled by backtracking.



Constants

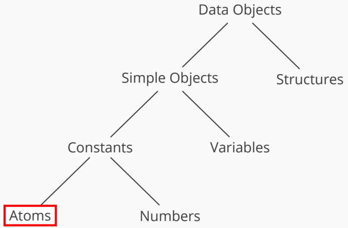
Constants represent a single element:
either an atom or a number.
Constants do not vary nor assume any value.



Atoms

Atoms are the fundamental building blocks of Prolog.

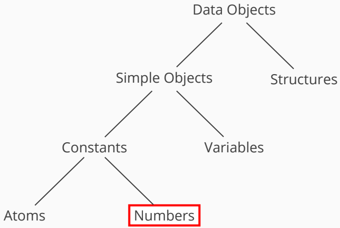
```
earth.
'air'.
fire.
water.
```



Numbers

Numbers in Prolog are either integers or floats.

```
0.3
2
94
```



Arithmetic

is

In Prolog we solve arithmetic operations via a special functor **is**.

is

In Prolog we solve arithmetic operations via a special functor **is**.

```
?- 3 is 2 + 1.
true.
```

is

In Prolog we solve arithmetic operations via a special functor **is**.

We can also solve equations.

```
?- 3 is 2 + 1.
true.
?- X is 8 / 3.
X = 2.6666666666666665.
```

is

In Prolog we solve arithmetic operations via a special functor **is**.

All the variables on the right hand side must be instantiated.

```
?- 3 is 2 + 1.
true.
?- X is 8 / 3.
X = 2.6666666666666665.
?- 3 is X + 1.
ERROR: Arguments are not sufficiently instantiated
```

is

In Prolog we solve arithmetic operations via a special functor **is**.

is is a special functor that calls an arithmetic solver - no matching.

```
?- 3 is 2 + 1.
true.
?- X is 8 / 3.
X = 2.6666666666666665.
?- 3 is X + 1.
ERROR: Arguments are not sufficiently instantiated
```

is

In Prolog we solve arithmetic operations via a special functor **is**.

Number are regular terms in Prolog.

```
?- 3 is 2 + 1.
true.
?- X is 8 / 3.
X = 2.6666666666666665.
?- 3 is X + 1.
ERROR: Arguments are not sufficiently instantiated
?- X = 3 + 2.
X = 3+2.
```

Regular Operations

Prolog uses standard symbols for arithmetic operations.

+ - * /

```
?- X is 3 + 2.
X = 5.
?- X is 3 - 2.
X = 1.
?- X is 3 * 2.
X = 6.
?- X is 3 / 2
X = 1.5.
```

```
?- X is div(3,2)
X = 1.
?- X is mod(3, 2)
X = 1.
```

Summary

Recap

Logical Connectives

- **:-** Implication — equivalent to if
- **,** Conjunction — equivalent to and
- **;** Disjunction — equivalent to or

Prolog Statements

Facts state what is always true.

```
man(socrates).
```

Rules state what is true if some conditions hold.

```
mortal(X) :- man(X).
```

Queries are how we interact with the program.

They allows us to check what logically follows from our facts and rules.

```
?- mortal(socrates).
```

Recap

Rules

Conjunction

```
head :- goal_1, goal_2, ..., goal_n.
```

Disjunction - Procedure

```
head :- goal_1; goal_2; ...; goal_n.
```

Disjunction is commonly written in different lines

```
head :- goal_1.
head :- goal_2.
...
head :- goal_n.
```

Recap

Matching

Two terms match if

- They are identical;
- Their variables can be instantiated so as to make them identical.

Search

- Prolog tries to match each goal in a query **in order**.
- If one of the goals fail, it backtracks to the last choicepoint.
- A choicepoint is a previous goal where more than one variable instantiation produce a match.
- The search process can be seen as a **tree**.