Introduction to AI with Prolog

Facts, Rules, and Queries

Alexey Sery
a.seryj@g.nsu.ru

Novosibirsk State University
*THE REAL SCIENCE

НГУ ФИТ

Department of Information Technologies
Novosibirsk State University

August 31, 2021

Tools and resources

We are going to use SWI-Prolog — a comprehensive free Prolog environment.

- You can download SWI-Prolog at this page:
  https://www.swi-prolog.org/download/stable
- Complete manual is available at
  https://www.swi-prolog.org/pldoc/doc_for?object=manual
- Online tool SWISH: https://swish.swi-prolog.org/
- Eclipse PDT — Prolog Development Tool:
  https://sewiki.iai.uni-bonn.de/research/pdt/docs/start
- Slides will be published weekly in the Google Classroom and on github:
  https://github.com/Inscriptor/IntroductionToAI

Work and grading policy

100% = 375 points

- 6 Assignments: 5% + 6% + 2 x 12% + 2 x 20%: 75%
- Exam: 40%
  - 5 = 150 pts, 4 = 100 pts, 3 = 50 pts, 2 = 0 pts
- Participation: 15%
- Tests: 5%
- Late day policy
  - 7 free late days; afterwards 10% off per day late
  - Assigments not accepted after 1 week late per assignment

Overview

Goals of the lesson

- Look into the syntax of Prolog language
- Define the main terms of Prolog: fact, rule, query, and knowledge base
- Define basic syntax units, such as atom, variables and term
- Look at some example programs and try to figure out what they do
- Get out feet wet with some programming

- **Facts**. Facts are used to state things that are unconditionally true in the domain of interest.
- **Rules**. Rules state information that is conditionally true of the domain of interest.

Knowledge Base (aka database) is a collection of facts and rules. Prolog programs are knowledge bases, collections of facts and rules which describe some collection of relationships that we find interesting. We use knowledge base by posing queries.

Queries. The Prolog interpreter responds to queries about the facts and rules represented in its database. In making a query you are asking Prolog whether it can prove that your query is true. If so, it answers *True* and displays any *variable bindings* that it made in coming up with the answer. If it fails to prove the query true, it answers *False*.

Example 1: Simple knowledge base

Knowledge base

```
kind(fruit, orange).
kind(fruit, apple).
kind(veg, tomato).
kind(veg, onion).
```

Queries

Simple Examples

Example 1: Simple knowledge base

Knowledge base

```
kind(fruit, orange).
kind(fruit, apple).
kind(veg, tomato).
kind(veg, onion).
```

Queries

```
?- kind(fruit, apple).  true.
?- kind(veg, apple).  false.
?- pred(veg, onion).  ERROR
```

Example 1: Simple knowledge base

Knowledge base

```
kind(fruit, orange).
kind(fruit, apple).
kind(veg, tomato).
kind(veg, onion).
color(red, apple).
color(red, tomato).
color(red, onion).
color(orng, orange)
```

Queries

```
?- kind(fruit, apple).  true.
?- kind(veg, apple).  false.
?- pred(veg, onion).  ERROR
```

Simple Examples

Example 1: Simple knowledge base

Knowledge base

```
kind(fruit, orange).
kind(fruit, apple).
kind(veg, tomato).
kind(veg, onion).
color(red, apple).
color(red, tomato).
color(red, onion).
color(orng, orange)
```

Queries

```
?- kind(fruit, apple).  true.
?- kind(veg, apple).  false.
?- pred(veg, onion).  ERROR
?- color(red, F). F = apple; F =
tomato; F = onion.
?- kind(fruit, F), color(red, F).
F = apple.
```

Example 2: Conjunctions and disjunctions in rules

Knowledge base

```
lovesMusic(anna).
lovesMusic(sam).
playsInstrument(anna).
musician(P) :- lovesMusic(P),
playsInstrument(P).
melomaniac(P) :- !,lovesMusic(P);
playsInstrument(P).
```

Queries

Example 2: Conjunctions and disjunctions in rules

Knowledge base

```
lovesMusic(anna).
lovesMusic(sam).
playsInstrument(anna).
musician(P) :- lovesMusic(P),
playsInstrument(P).
melomaniac(P) :- !,lovesMusic(P);
playsInstrument(P).
```

Queries

```
?- musician(anna).
true.
?- musician(sam).
false.
?- melomaniac(anna).
true.
?- melomaniac(sam).
true.
?- melomaniac(Person).
Person=sam; Person=anna.
```

Example 3: Negations

Knowledge base

```
cat(fluffy).
cat(cornie).
bird(butch).
dog(bayley).
good(fluffy).
hasClaws(X) :- cat(X).
hasClaws(X) :- bird(X).
hasClaws(X) :- not(dog(X)).
animal(X) :- cat(X);bird(X);dog(X).
domestic(X) :-
animal(X),not(hasClaws(X)).
domestic(X) :- animal(X),good(X).
```

Queries

Example 3: Negations

Knowledge base

```
cat(fluffy).
cat(cornie).
bird(butch).
dog(bayley).
good(fluffy).
hasClaws(X) :- cat(X).
hasClaws(X) :- bird(X).
hasClaws(X) :- not(dog(X)).
animal(X) :- cat(X);bird(X);dog(X).
domestic(X) :-
animal(X),not(hasClaws(X)).
domestic(X) :- animal(X),good(X).
```

Queries

```
?- domestic(bayley).
true.
?- domestic(cornie).
false.
?- cat(C),domestic(C)
C = fluffy.
?- cat(C),not(domestic(C))
C = cornie.
```

Facts, rules and queries in Prolog are built of terms

- Atoms
- Numbers
- Strings
- Variables
- Compex terms — structures

Atoms

1. A string of characters made up of upper-case letters, lower-case letters, digits, and underscore characters, that begins with a lower-case letter.
2. An arbitrary sequence of characters enclosed in single quotes.
3. A sequence of special characters.

Example

```
chuck_norris, bayley, cornie, someCamelCaseStringNumber1, 'Chuck
Norris', 'Some arbitrary string', '@!?@', =====>, :-, @>
```

Numbers

1. **Real numbers**, though not particularly important in typical applications, are supported in Prolog.

Example

2.71828, 82.19284, $\pi, e, \ldots$

2. **Integers** are useful for many tasks, such as counting the elements of a list.

Example

$-2, -1, 0, 1, 2, 3, \ldots$

Variables and strings

1. A variable is a sequence of upper-case letters, lower-case letters, digits and underscore character that starts either with an upper-case letter or with underscore.

Example

```
X, Y, Var, Chuck_Norris, _someVar, _1zx, _
```

2. String is an arbitrary sequence of characters enclosed in double quotes.

Example

"Some arbitrary string line"

Complex terms

- Complex terms are built out of **functor (predicate)** followed by a sequence of **arguments**.
- The arguments are put in parentheses and are separated by commas.
- The functor of a term **must** be an atom.
- Arguments can be any kind of term.
- The number of arguments that a complex term has is called its **arity**.

Complex terms

- Any **constant** is a term. Constants are atoms, numbers or strings.
- Any **variable** is a term.
- Any sequence of a form `f(a1, a2, ...)` where `f` is an atom, and `a1`, `a2`, ... are terms, is a term.
- Conjunction and disjunction of terms are terms: `(T1, T2)`, `(T1; T2)`.
- There are no other terms.

Example

`g(f(x, y))`

Prolog Syntax

Which of the character sequences are atoms, variables or neither of them?

1. `vARIABLE`
2. `Variable`
3. `x`
4. `XY1`
5. `chuck_norris_tells_simon_what_to_do`
6. `_john`
7. `'_jonh'`
8. `'John likes everybody'`
9. `Chuck Norris plays russian roulette with a fully loded revolver and wins`

Prolog Syntax

Which of the sequences are terms, and which are not. For every term indicate its functor and arity.

1. `loves(vincent,mia)`
2. `'loves(vincent,mia)'`
3. `Eats(cat,mouse)`
4. `hasChildren(cat,kittens)`
5. `and(musician(jody),artist(mia))`
6. `and(musician(X),artist(Y))`
7. `_and(musician(jody),artist(mia))`
8. `(Butch kills Vincent)`
9. `kills(Butch,Vincent)`

How many facts, rules, clauses and predicates there are in the knowledge base?

```
cat(fluffy).
cat(cornie).
bird(butch).
dog(bayley).
good(fluffy).
hasClaws(X) :- cat(X).
hasClaws(X) :- bird(X).
hasClaws(X) :- not(dog(X)).
animal(X) :- cat(X);bird(X);dog(X).
domestic(X) :- animal(X),not(hasClaws(X)).
domestic(X) :- animal(X),good(X).
```