Programming Paradigms CSI2120 – Winter 2018

Jochen Lang

EECS, University of Ottawa

Canada

uOttawa

L'Université canadienne
Canada's university

Université d'Ottawa | University of Ottawa

uOttawa.ca

Logic Programming in Prolog

- **History**
- **Logic Programming**
- **Prolog**
  - facts and rules
  - atoms and variables
- **Queries**
  - Search
  - Variable instantiation
  - Unification
- **First Examples**

Prolog History

- **Paradigm: declarative, logic programming**

- **1972: A. Colmerauer and P. Roussel, Marseille, created the language**
  - Envisioned application was natural language processing
- **1977: First compiler by D.H. Warren, Edinburgh**
- **1980: Borland Turbo Prolog**
- **1995: ISO Prolog**

The Birth of Prolog

- **User:** Cats kill mice. Tom is a cat who does not like mice who eat cheese. Jerry is a mouse who eats cheese. Max is not a mouse. What does Tom do?
- **Computer:** Tom does not like mice who eat cheese. Tom kills mice.
- **User:** Who is a cat?
- **Computer:** Tom.
- **User:** What does Jerry eat?
- **Computer:** Cheese.
- **User:** Who does not like mice who eat cheese?
- **Computer:** Tom.
- **User:** What does Tom eat?
- **Computer:** What cats who do not like mice who eat cheese eat.

Alain Colmerauer and Philippe Roussel. The birth of Prolog. In History of programming languages---II, ACM, New York, NY, USA 331-367, 1996.

Basis of Conversation

- **The logical formulas created made use of:**
- **constants representing elements**
  - Tom, Jerry, Max, Cheese
- **constants representing sets,**
  - Cats, Mice, MiceWhoEatCheese, CatsWhoDoNotLikeMiceWhoEatCheese;
- **constants representing binary relations between sets,**
  - Kill, DoesNotLike, Eat;
- **a functional symbol of arity 1 and two relational symbols of arity 2 and 3,**
  - The, Subset, True.

Alain Colmerauer and Philippe Roussel. The birth of Prolog. In History of programming languages---II, ACM, New York, NY, USA 331-367, 1996.

Logical Clauses relating the Symbols

$$(\forall \mathrm{x})[\mathrm{Subset}(\mathrm{x},\mathrm{x})],$$
$$(\forall x)(\forall y)(\forall z)[Subset(x,y) \land Subset(y,z) \rightarrow Subset(x,z)],$$
$$(\forall \mathrm{a})(\forall b)[Subset(The(a),The(b)) \rightarrow Subset(The(b),The(a))],$$
$$(\forall \mathrm{x})(\forall y)(\forall \mathrm{r})(\forall x')(\forall \mathrm{y}')$$
$$[True(r,x,y) \land Subset(x,x') \land Subset(y,y') \rightarrow True(r,x',y')].$$

$The(\,)$ is a set with a single element.

Alain Colmerauer and Philippe Roussel. The birth of Prolog. In History of programming languages---II, ACM, New York, NY, USA 331-367, 1996.

Applications

- **Applications of declarative, logic programming:**
  – symbolic computation (i.e. non-numeric)
- **Symbolic computation applications include:**
  – Many areas of artificial intelligence (property of declarative)
  – Understanding natural language (specific to logic programming)
  – Relational databases
  – Mathematical logic
  – Abstract problem solving
  – Design automation
  – Symbolic equation solving
  – Biochemical structure analysis

Programming in Prolog

Prolog is *descriptive* (as opposed to *prescriptive*)

- **descriptive: describing known *facts* and *relationships* (or rules) about a**
  - specific problem
- **as opposed to**
  - prescriptive: prescribing the sequence of steps taken by a computer to solve a specific problem

Programming Steps in Prolog

- **Specify Facts**
  - which are true in a problem domain. Will remain true forever.
- **Define rules**
  - which when applied establish new facts.
- **Start queries**
  - and the prolog interpreter answers
- **Prolog uses first order logic to prove answers**
  - It answers Yes following a successfully proven answer
  - It answers No otherwise
    - A no answer means it could not prove a positive answer

First Order Logic

- **Consists of**
  - predicate symbols
  - equality
  - negation
  - logic binary connections
  - quantifiers ‘for all …’ and ‘there exists … such that’
- **More on this later …**

Computation in Prolog

Specified by

- **partly by the logical declarative semantics of Prolog (more on this later),**
- **partly by what new facts Prolog can infer from the given ones, and**
- **partly by explicit control information supplied by the programmer.**
  - In other words Prolog has/requires some imperative, or prescriptive features.

Facts

Example: “Dogs like cats” with individuals “dogs”, “cats” and relationship “like”

In Prolog: `like(dogs,cats).`

- **lower case for both individuals and relationships**
- **relationship (or predicate) is written first**
- **individuals (or arguments) are written in parenthesis, separated by commas**
- **ends with a dot “.”**
- **order of arguments is important, in this case “liker” is first, “liked” is second, i.e.,** `like(cats,dogs).` **is a different fact.**

More facts

Other examples:

```
domestic(cows).          % cows are domestic animals.
faster(horses,cows).     % horses run faster than cows
take(cats,milk,cows).    % cats take milk from cows
isYellow(hay).           % hay is yellow.
eat(cows,hay).           % Cows eat hay.
```

- **Constants or Atoms**
  - Example: `cows, horses, hay, cats, milk`
  - Symbolic: small caps letter followed by letters and numbers
  - Numbers : integer and float

Interpretation of Facts

Is “cats” an individual?

Yes, but there is more than one way to interpret it.

- **a particular type of cat, e.g., house cats**
- **a family of animals encompassing tigers, leopards, etc.**

Either interpretation is fine. The program context will need to define which one is meant.

- **If a program needs more than one interpretation then the names of the individuals have to be different, e.g.,**
  – houseCats and catsFamily

More on Facts

Arity of Predicates

Predicates can have an arbitrary number of arguments

```
domestic/1 isYellow/1  % 1 argument
faster/2  like/2  eat/2 % 2 arguments
takes/3 % 3 arguments
```

Facts that are false in the real world can be used.

- **faster(snails,cheetahs).**

Database

- **a collection of facts (part of a program)**

Queries or Questions

Questions are about individuals and their relationships

Example: `?- eat(cats,mice).`

- **Means "Do cats eat mice?" or "Is it a fact that cats eat mice?"**
- **Note as before, cats are interpreted as a specific species (house cats) and mice are all type of mice.**
- **Note that the syntax is the same as for facts, except for the special symbol `?-` (printed by the interpreter) to distinguish from a fact.**

A Database

```
like(horses,fish).
like(dogs,cats).
like(cats,mice).
like(dogs,mice).
like(horses,racing).
like(cats,horses).
like(tigers,cats).
like(cats,hay).
like(cows,grass).
like(cows,hay).
like(horses,hay).
```

Simple Queries

```
?- like(dogs,bones).
?- like(cats,dogs).
?- like(cats,hay).
?- enjoy(horses,racing).
```

Variables

More interesting questions of the type: "Do cats like X?"

- We want Prolog to tell us what X could stand for.
- Prolog searches through all the facts to find things cats like.

- **In Prolog** `?- like(cats,X).`
  - Variables start with uppercase letters.

How Prolog Answers

- When Prolog is first asked this question, variable X is initially not instantiated.
- Prolog searches through the database, looking for a fact that *unifies* with the question (or *query* or *goal*).
- If there is an *uninstantiated* variable as argument, Prolog searches for any fact where the predicate is “like” and the first argument is “cats”.
- When such a fact is found, X becomes *instantiated* with the second argument of the fact.
- Prolog searches the facts *in order* (top to bottom).
- X is first *instantiated* to “mice".
- Prolog marks the place in the database where the *unifier* is found.

Multiple Answers

- **When entering ; we ask Prolog to re-satisfy the goal**
  - or to search for another solution

- **Prolog resumes its search, starting from where it left the place-marker.**
- **We are asking Prolog to re-satisfy the question, and resume search with X *uninstantiated* again.**
- **After a ; false means "no more answers"**

Conjunctions

"Do cats and dogs like each other?"

```
?- like(cats,dogs), like(dogs,cats).
```

Note

- **`,` represents "and"**
- **can have any number of questions separated by `,` (comma) and ending with `.` (dot)**

Example with Variables

“Is there anything that horses and cows both like?”

2 steps:

1. Find out if there is some X that cows like.
2. Then find out if horses like whatever X is.

```
?- like(cows,X), like(horses,X).
```

Note:

- **After finding the first answer for X (hay), Prolog marks the place in the database.**
- **Prolog attempts to satisfy the second goal (with X instantiated).**
- **If it succeeds, Prolog marks (separately) that goal's place in the database.**
- **Each goal keeps its own place-marker.**

Rules

- **A *rule* is a general statement about objects and their relationships.**
  - “Horses like any type of animal who likes hay.” or, in other words
  - “Horses like X if X like hay.”

```
likes(horses,X) :- like(X,hay).
```

Note:

- **A Prolog rule has a head and body, separated by ":-" pronounced “if”.**
- **The head is on the left; the body is on the right.**
- **A rule ends in "."**

Rules

- **The head of the rule describes what fact the rule is intended to define.**
- **The body can be a conjunction of goals.**
  - “Horses like X if X like hay and mice."

```
like(horses,X) :- like(X,hay), like(X,mice).
```

- **There are 3 occurrences of X. Whenever X becomes instantiated, all X's are instantiated to the same thing.**

Summary

- **Logic Programming**
- **Prolog**
  - facts and rules
  - atoms and variables
- **Queries**
  - Search
  - Variable instantiation
  - Unification
- **First Examples**