# Logic Programming with Prolog

## Concepts of Programming Languages

## 14 December 2020

Bernhard Saumweber
Rosenheim Technical University

# Prolog

- « *programmation en logique* »
- Developed by French programmer Alain Colmerauer (1941-2017)
- First version from 1972
- Turing complete
- Programs consist of a knowledge base containing facts and rules
- You cannot give "steps" how to solve the problem
- User posts a goal (query)
- Prolog interpreter uses rules and facts to find a solution

## Terms

The single data type in prolog is a **term**. Terms can be either

- Atoms
- Numbers
- Variables
- Compound terms

## Atoms

- Atoms are general purpose names without any inherent meaning
- Start with a lower case letter or are quoted

Examples:

- x
- red
- romeo
- pizza
- 'Pizza'
- 'foo bar'

## Variables

- Variables are placeholders for arbitrary terms
- Can be bound in terms
- Start with an upper case letter or an underscore

Examples:

- X
- Foo
- Romeo

## Facts

Clauses without body are called **facts**.

The fact "Romeo loves Juliet" can be expressed as:

```
loves(romeo, juliet).
```

where `loves` is a **predicate** and `romeo` and `juliet` are called **atoms**.
Facts end with a dot (just like a semicolon in Java).

## Rules

Clauses with a body are called **rules** and expressed as Horn clauses.

The rule "Juliet loves Romeo as long as Romeo loves Juliet." can be expressed as:

```
loves(juliet, romeo) :- loves(romeo, juliet).
```

- :- denotes logic implication (similar to an `if`)
- Left side is true if right side is true
- You can read it as an reversed implication arrow

Rules end with a dot (just like a semicolon in Java).

# Facts are Rules

The fact from the previous slide

```
loves(romeo, juliet).
```

can also be written as a rule:

```
loves(romeo, juliet) :- true.
```

## Loading a knowledge base

In Prolog, loading code is referred to as **consulting**.

One can load a knowledge base by either running (here shown with SWI Prolog):

```
swipl filename.pl
```

Or by loading interactively using:

```
[filename].
```

## Interactive Prompt

- Prolog can be used interactively by entering queries at the Prolog prompt ?-
- If there is no solution, Prolog writes no or `false`
- If a solution exists then it is printed, followed by yes or `true`
- If there are multiple solutions to the query, then these can be requested by entering a semicolon ; or n

## Conjunction and Disjunction

Multiple predicates in the body of an expression can be composed as

- conjunction (logical and) using , (comma)
- disjunction (logical or) using ; (semicolon)

Note: usually it is better to write separate rules instead of a disjunction.

Example:

```
happy(juliet).
with_romeo(juliet).
dances(juliet) :- happy(juliet), with_romeo(juliet).
```

# Playing with variables

Given some facts:

```
loves(romeo, juliet).
loves(william, juliet).
```

Variables can be used to get everyone that loves Juliet:

```
?- loves(X, juliet).
X = romeo ;
X = william.
```

Or to express general rules:

```
grandparent(X, Y) :-
    parent(X, Z),
    parent(Z, Y).
```

# Exercise 1

## Naming Conventions

- When referring to facts, the notion `loves/2` refers to the predicate `loves` with two arguments
- More examples: `true/0`, `length/2`, `,/2`, `call/1`, `call/2`, ...
- It is possible to define facts with a different number of arguments (differnt **arity**)

# Hello World

Prolog supports printing values:

```
write('Hello World!'), nl.
```

Can also be used in rules for printing intermediate results.

Or to print the actual results in a nicer way:

```
get_loves_juliet :-
    loves(X, juliet),
    write(X), write(' loves Juliet'), nl.

?- get_loves_juliet.
romeo loves Juliet ;
william loves Juliet .
```

There is also `format/2`.

## Atoms vs Strings

Atoms do not have quotes or are single quoted:

```
bob
'this is an atom'
'AlsoAtomNotVariable'
```

Strings are double quoted (only in modern Prolog interpreters):

```
"this is a string"
```

Usually it is fine to use atoms only. However, they have restrictions on length.

# Lists

Denoted in brackets

```
L = [a,b,c].
```

`length/2` gives you the length of a list:

```
?- length([1,2,3], X).
X = 3.
```

Split into head and tail:

```
?- [Head|Tail] = [1,2,a].
Head = 1,
Tail = [2, a].
```

There is also `reverse/2`, append/2 and many more.

SWI Prolog built-in list operations (https://www.swi-prolog.org/pldoc/man?section=builtinlist)

# Arithmetic

- Standard operators +, -, *, /, >, <, =<, >=, =\=, =:= and `is` can be used
- Support for rational numbers as well as floats

Examples:

```
?- X is 1, Y is X+1.
X = 1,
Y = 2.

?- 3 =< 5.
true.

get_age(BirthYear) :-
    Age is 2020-BirthYear,
    format('Someone born in ~w is age ~w in ~w.~n', [Age, BirthYear, 2020]).
```

SWI Prolog General purpose arithmetic (https://www.swi-prolog.org/pldoc/man?section=arithpreds)

## Difference between is, =, == and =:=

Number **is** Expr evaluates to true when Number is the **value** to which Expr evaluates

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

Term1 = Term2 is true if Term1 **unifies** with Term2

```
?- 1 = 1+1-1.
false.
```

Term1 == Term2 is true if Term1 is **equivalent** to Term2

```
?- 1 == 1+1-1.
false.
```

Expr1 =:= Expr2 is to true if expression Expr1 evaluates to a number **equal** to Expr2, i.e. float 1.0 =:= integer 1.

```
?- 1 =:= 1+1-1.
true.
```

# Unification

Unification is mostly done implicitly while matching the head of a predicate. However, it is also provided by the `=/2` predicate that tests if its two arguments **unify**. It is usually written infix.

Two terms unify if they are the same term or if they contain variables that can be uniformly instantiated with terms in such a way that the resulting terms are equal.

```
?- bob = bob.
true.

?- X = 2.
X = 2.

?- 2 = X.
X = 2.

?- X = 2*3.
X = 2*3.

?- loves(X,X) = loves(romeo,juliet).
false.
```

# Exercise 2

# Not in Prolog

\+ Goal (same as `not/1`) is true if Goal *cannot be proven*.

```
?- \+ (2 = 4).
true.
```

This is **not** the same as logical not!!

Logical negation in Prolog is problematic:

- The only way to tell if a proposition is false is to prove it.
- If the attempt fails, Prolog concludes that the proposition is false.
- This may take very long or forever if not all relevant facts or rules have been given.
- This concept is known as *negation as failure*.

Try to avoid negative terms. Think positive!

# Solution by side effects

Sometimes `write` can be used for printing the actual solution as a side effect:

```
% Towers of Hanoi

move(1,X,Y,_) :- write('Move top disk from '), write(X), write(' to '), write(Y), nl.
move(N,X,Y,Z) :-
    N > 1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).

run :- move(3,left,right,center).
```

Loops are done with recursion just like in functional programming.

# Debugging Tools

Trace/1 enables tracing of a given predicate, very useful for debugging:

```
trace(loves).
```

Trace/0 traces everything:

```
trace.
```

Listing gives you a list of all known facts:

```
listing(loves).
```

You may also simply print intermediate results:

```
loves(X, Y) :- write(X), write(Y), nl.
```

## Execution

- Logically, the Prolog engine tries to find a resolution refutation of the negated query.
- The resolution method used by Prolog is called SLD resolution.
- If the negated query can be refuted, it follows that the query, with the appropriate variable bindings in place, is a logical consequence of the program.
- In that case, all generated variable bindings are reported to the user, and the query is said to have succeeded.
- If there is no solution, Prolog writes no or `false`

## Execution

- Operationally, Prolog's execution strategy can be thought of as a generalization of function calls in other languages, one difference being that multiple clause heads can match a given call.
- In that case, the system creates a choice-point, unifies the goal with the clause head of the first alternative, and continues with the goals of that first alternative.
- If any goal fails in the course of executing the program, all variable bindings that were made since the most recent choice-point was created are undone, and execution continues with the next alternative of that choice-point.
- This execution strategy is called chronological **backtracking**.

## Live Coding

Three missionaries and three cannibals want to cross a river with a boat.
However, the small boat only carries at most two people.
On both banks, the missionaries may not be outnumbered by the cannibals, otherwise it is to be feared that the cannibals overpower the missionaries and eat them alive.
Find a solution for the six people to safely cross the river.

# Exercise 3

# There is more to Prolog

- IO, Structures, Meta Predicates, Unit Tests, ...
- Tail Recursion, Hashing, Tabling, ...
- Many libraries, e.g. Unicode, YAML, HTTP, C-Bindings, ...
- Paxos: replicating key-value store (remember it from distributed computing?)

SWI Prolog Manual (https://www.swi-prolog.org/pldoc/man?section=builtin)

# Thank you

Bernhard Saumweber
Rosenheim Technical University
bernhard.saumweber@qaware.de (mailto:bernhard.saumweber@qaware.de)
http://www.qaware.de (http://www.qaware.de)