Programming Language Concepts
Logic Programming – Prolog

Janyl Jumadinova
14 March, 2017
Japan’s “Fifth Generation” project–1982:

(https://en.wikipedia.org/wiki/Fifth_generation_computer)
“These Fifth Generation computers will be built around the concepts of logic programming.”
Prolog Background

Japan’s “Fifth Generation” project–1982:

(https://en.wikipedia.org/wiki/Fifth_generation_computer)
“These Fifth Generation computers will be built around the concepts of logic programming.”

- The use of logic to express information in a computer.
- The use of logic to present problems to a computer.
- The use of logical inference to solve these problems.
The Fifth Generation Project

“The project imagined a parallel processing computer running on top of massive databases (as opposed to a traditional filesystem) using a logic programming language to define and access the data.”
The Fifth Generation Project

- “The project imagined a **parallel processing** computer running on top of massive **databases** (as opposed to a traditional filesystem) using a **logic programming language** to define and access the data.”

- Depending on who you ask, the Fifth Generation project was either “Ahead of its time” or was a failure.
Prolog

- Program consists of a database of **facts** and a set of **rules**.
Program consists of a database of **facts** and a set of **rules**.

- Facts are expressed as “predicates”—the programmer supplies the meaning.
Prolog

- Program consists of a database of **facts** and a set of **rules**.
- Facts are expressed as “predicates”–the programmer supplies the meaning.

parent(hank,ben). % "hank is a parent of ben"
isa(swan,bird).   % "a swan is a bird"
required(cs111). % "cs111 is required"
prereq(cs111,cs112).
eats(unicorn,rose).
stooges(moe,larry,curly).
Prolog

- Constants ("atoms" and names of predicates) begin with lowercase letters; variables are capitalized.

- Rules specify conditions that must hold for a predicate to be true:
  
  \[ \text{grandparent}(X,Y) \text{ :- } \text{parent}(X,Z), \text{parent}(Z,Y). \]

- This means "\[ X \] is a grandparent of \[ Y \] if there exists a \[ Z \] such that \[ X \] is a parent of \[ Z \] and \[ Z \] is a parent of \[ Y \]." The symbol \[ :- \] should be read as "if" and a comma should be read as "and."
Prolog

- Constants ("atoms" and names of predicates) begin with lowercase letters; variables are capitalized.
- Rules specify conditions that must hold for a predicate to be true:

```prolog
grandparent(X,Y) :- parent(X,Z),parent(Z,Y).
```
- This means "X is a grandparent of Y if there exists a Z such that X is a parent of Z and Z is a parent of Y." The symbol "\(\text{:-}\)" should be read as "if" and a comma should be read as "and."
Constants ("atoms" and names of predicates) begin with lowercase letters; variables are capitalized.

Rules specify conditions that must hold for a predicate to be true:

\[
\text{grandparent}(X,Y) :- \text{parent}(X,Z), \text{parent}(Z,Y).
\]

This means "\(X\) is a grandparent of \(Y\) if there exists a \(Z\) such that \(X\) is a parent of \(Z\) and \(Z\) is a parent of \(Y\)." The symbol "\(:-\)" should be read as "if" and a comma should be read as "and".
Prolog

- A program is more like a database of facts and rules.
- We solve problems by querying this database.
Prolog

- A program is more like a database of facts and rules.
- We solve problems by querying this database.

beats(scissors, paper).
beats(paper, rock).
beats(rock, lizard).
beats(lizard, spock). throws(sheldon, spock).
beats(spock, scissors). throws(leonard, lizard).
beats(scissors, lizard). throws(bernie, paper).
beats(lizard, paper). throws(amy, rock).
beats(paper, spock). throws(howard, scissors).
beats(spock, rock). wins(X, Y):-
  beats(rock, scissors). throws(X, R), throws(Y, S), beats(R, S).
The last item is a rule:

\[\text{wins}(X,Y) :\text{throws}(X,R),\text{throws}(Y,S),\text{beats}(R,S)\]

It should be read as:

"\(X\) wins over \(Y\) if there exist values \(R\) and \(S\) such that..."
The last item is a rule:

\[
\text{wins}(X,Y) :- \text{throws}(X,R), \text{throws}(Y,S), \text{beats}(R,S).
\]

It should be read as:

“X wins over Y if, for some values of R and S, X throws R, Y throws S, and R beats S.”

“X wins over Y if there exist values R and S such that...”
Prolog

$ gplog
| ?- [facts].
(1 ms) yes
| ?- wins(X,Y).
X = sheldon
Y = amy ? ;
X = sheldon
Y = howard ? ;
X = leonard
Y = sheldon ? ;
...

Consult a database named “facts.pl” (ordinary text file in local directory)

Pose a query: “For what values of X and Y does X win over Y?”

System responds with candidate values for variables X and Y

Each time “;” is entered, a new search is made; when no more solutions are found, system says “no”
How does it work?

- Prolog tries to match the pattern of the query with one of the facts or with the left-hand side of one of the rules. Example: “wins(X,Y)” matches the pattern of the left-hand side of rule

```
wins(X,Y):- throws(X,R),throws(Y,S),beats(R,S).
```

- If a fact is found, we are done
- Otherwise we recursively query each of the terms in the right-hand side of the rule: “throws(X,R)” and “throws(Y,S)” BOTH match the fact “throws(sheldon,spock)”, but there is no match for “beats(spock, spock)”, so we backtrack to find more matches...
Prolog

How does it work?

- Prolog tries to match the pattern of the query with one of the facts or with the left-hand side of one of the rules. Example: “wins(X,Y)” matches the pattern of the left-hand side of rule

  \[ \text{wins}(X,Y) : - \text{throws}(X,R), \text{throws}(Y,S), \text{beats}(R,S). \]
How does it work?

- Prolog tries to match the pattern of the query with one of the facts or with the left-hand side of one of the rules. Example: “wins(X,Y)” matches the pattern of the left-hand side of rule
  ```prolog
  wins(X,Y):- throws(X,R),throws(Y,S),beats(R,S).
  ```
- If a fact is found, we are done
How does it work?

- Prolog tries to match the pattern of the query with one of the facts or with the left-hand side of one of the rules. Example: “wins(X,Y)” matches the pattern of the left-hand side of rule

  \[
  \text{wins}(X,Y) :\neg \text{throws}(X,R), \text{throws}(Y,S), \text{beats}(R,S).
  \]

- If a fact is found, we are done

- Otherwise we recursively query each of the terms in the right-hand side of the rule:

  “throws(X,R)” and “throws(Y,S)” BOTH match the fact “throws (sheldon, spock)”, but there is no match for “beats(spock, spock)”, so we backtrack to find more matches...
... and eventually we find a match-up:

\[
\text{throws}(X,R), \quad \text{throws}(Y,S), \quad \text{beats}(R,S)
\]

\[
\text{throws}(\text{sheldon}, \text{spock})
\]

\[
\text{throws}(\text{howard}, \text{scissors})
\]

\[
\text{beats}(\text{spock}, \text{scissors})
\]

When a match is made that involves a variable, a BUNDLING occurs between the variable and the matched item. So, \(X = \text{sheldon}\), \(R = \text{spock}\), \(Y = \text{howard}\), \(S = \text{scissors}\). Bindings must be consistent.