In Text: Chapter 16

Prolog

- A logic programming language
- Prolog programs consist of collections of statements
- There are only a few kinds of statements in Prolog, but they can be complex
  - Fact statements, rule statements, and goal statements
- All Prolog statements are constructed from terms

Fact Statements

- Correspond to Headless Horn clauses
- Fact statements are propositions that are assumed to be true, and from which new information can be inferred
- E.g., female(shelley).
  female(mary).
  mother(mary, shelley).

Rule Statements

- Correspond to Headed Horn clauses
- They describe implication rules between propositions, or logical relationship between them: if a set of given conditions are satisfied, what conclusion can be drawn
- The consequent of a statement is a single term, while the antecedent can be either a single term or conjunction

Conjunctions

- The AND operation in conjunctions is implied in Prolog
- The structures that specify atomic propositions in a conjunction are separated by commas
- The commas can be considered as AND operators

Rule Statements

- E.g., grandparent(X, Z) :- parent(X, Y),
  parent(Y, Z),
  where X, Y, and Z are universal objects
  - It states that if there are instantiations of X, Y, and Z such that parent (X, Y) is true, and parent (Y, Z) is true, then for those same instantiations of X, Y, and Z, grandparent (X, Z) is true
Goal Statements

- Also correspond to Headless Horn clauses
- Goal statements are propositions describing the theorem that we want the system to either prove or disprove
  - E.g., man(fred)
- Because goal statements and some nongoal statements have the same form, a Prolog implementation must have some means to distinguish between the two

Goal Statement

- If we want to find all possible solutions, we can ask the interpreter to continue by typing a semicolon:
  - C = seattle ;
  - C = rochester.

Another Example

- Can we define propositions in the following way?
  - takes(jane_doe, his201).
  - takes(jane_doe, cs254).
  - takes(ajit_chandra, art302).
  - takes(ajit_chandra, cs254).
  - classmates(X, Y) :- takes(X, Z), takes(Y, Z).
  - What does the following query return?
    - ?- classmates(jane_doe, X).

  - X = jane_doe ;
  - X = jane_doe;
  - X = ajit_chandra.
  - How should we modify the rule so that the student is not considered as a classmate of himself or herself?
    - classmates(X, Y) :- takes(X, Z), takes(Y, Z), X \= Y.
Prolog Programs

- **ASSERT** (define)
  - FACTS about OBJECTS
  - RULES (“CLAUSES”) that inter-relate facts
- Ask QUESTIONS about objects and their relationship
  - GOALS

Some Prolog FACTS

```
?- (assert (father (michael, cathy))).
?- (assert (father (chuck, michael))).
?- (assert (father (chuck, julie))).
?- (assert (father (david, chuck))).
?- (assert (father (sam, melody))).
?- (assert (mother (cathy, melody))).
?- (assert (mother (hazel, michael))).
?- (assert (mother (hazel, julie))).
?- (assert (mother (melody, sandy))).
?- (assert (made_of (moon, green_cheese))).
```

Some Prolog RULES

- A person’s parent is their mother or father
  `?- (assert ((parent(X, Y) :- father(X, Y); mother(X, Y))))).
- A person’s grandfather is the father of one of their parents
  `?- (assert ((grandfather(X,Y) :- father(X, A), parent(A, Y))))`.

Some Prolog QUESTIONS

- Is chuck the parent of julie?
  `?- parent(chuck, julie).
- Is john the father of cathy?
  `?- father(john, cathy).

Note:
- No “assert”s
- No use of variables

Prolog Notes

- **atoms**: symbolic values of Prolog
  - father (bill, mike)
  - Strings of letters, digits, and underscores starting with a **lower case** letter
- **variable**: unbound entity
  - father (X, mike)
  - Strings of letters, digits, and underscores starting with an **upper case** letter
  - Variables are **not** bound to type by declaration

Prolog Notes

- **FACTS**: UNCONDITIONAL ASSERTIONS OF “TRUTH”
  - `(assert(mother(carol, jim))).`
    - assumed to be true
    - contains no variables
    - stored in database
Prolog Notes

• RULES: ASSERTIONS from which conclusions can be drawn if given conditions are true
  \[ \text{assert}(\text{parent}(X, Y) \iff \text{father}(X, Y); \text{mother}(X, Y)). \]
  – contains variables for instantiation
  – also stored in database

An Example

FACTS

| ?- (assert(color(banana, yellow))).
| ?- (assert(color(squash, yellow))).
| ?- (assert(color(apple, green))).
| ?- (assert(color(peas, green))).

bob eats green colored vegetables

RULE
| ?- (assert((eats(bob, X) \iff color(X, green), vegetable(X)))).

An Example

What does bob eat?

| ?- eats(bob, X).
| color(banana, green) \iff no
| color(squash, green) \iff false
| color(apple, green) \iff yes
| vegetable(apple) \iff no
| color(peas, green) \iff yes
| vegetable(peas) \iff yes

Therefore X = peas

Does bob eat apples?

| ?- eats(bob, apple).
| color(apple, green) \iff match
| vegetable(apple) \iff false

false

false

Does bob eat squash?

| ?- eats(bob, squash).
| color(squash, green) \iff no

false

Instantiation & Unification

FACTS

\[ \text{assert} \text{color(apple, red)).} \]
\[ \text{assert} \text{color(banana, yellow)).} \]
\[ \text{color}(X, \text{yellow}). \]
\[ \text{X = apple} \quad \text{color}(X, \text{red)).} \]
\[ \text{X = banana} \quad \text{color}(X, \text{yellow)).} \]
\[ \text{X = apple} \quad \text{color}(X, \text{red)).} \]
\[ \text{X = banana} \quad \text{color}(X, \text{yellow)).} \]

Ask the question (goal):

Does there exist (or: Give me) an X such that X is the color yellow

instantiation no matching pattern

X = banana color(banana, yellow)

instantiation match

X = banana results in match of goal with database item

Prolog Notes

• DISJUNCTIVE RULES: X if Y or Z
  \[ \text{assert}((\text{parent}(X, Y) \iff \text{father}(X, Y))). \]
  \[ \text{assert}((\text{parent}(X, Y) \iff \text{mother}(X, Y))). \]
  or
  \[ \text{assert}((\text{parent}(X, Y) \iff \text{father}(X, Y); \text{mother}(X, Y))). \]
Prolog Notes

- **CONJUNCTIVE RULES**: X if Y AND Z
  
  (assert((father(X, Y) :- parent(X, Y), male(X)))).

- **NEGATION RULES**: X if Not Y
  
  (assert((good(X) :- not(bad(X)))).
  (assert((mother(X, Y) :- parent(X, Y), not(male(X)))).

- When we ask a query that will result in TRUE, we get the right answer:
  ?- older(george, mary).
  yes

- When we ask a query that will result in FALSE, we get into an endless loop
  ?- older(mary, john).

Left Recursion Problem

- The first element in older is the predicate that is repeatedly tried
- To solve the problem, remove the older rule and replace with:
  is_older(X, Y) :- older(X, Y).
  is_older(X, Z) :- older(X, Y), is_older(Y, Z).
- Now:
  ?- is_older(mary, john).
  false

Prolog Notes

- Prolog is more than "LOGIC"
  - Math
  - List manipulation

Consult File Format

[x]. or consult(x).

- File x.pl:
  husband(tommy, claudia).
  husband(mike, effie).
  mother(claudia, sannon).
  mother(effie, jamie).
  father(X, Y) :- mother(W, Y), husband(X, W).
  parent(X, Y) :- father(X, Y); mother(X, Y).
- Note: No assert's, but can still state
  Facts and Rules
Consult File

- Cannot state question/goal in a consult file

    ?- consult(x).

Suggested Approach to Specifying Solution

- Use a consult file to define facts and rules
  - Instantiate prolog
  - "consult" file interactively
  - Interactively ask questions to see if facts/rules yield expected results
  - Change consult as needed
    - Need to reinitiate prolog and re"consult"

Suggested Approach to Specifying Solution

- Construct I/O redirected file to include
  - Consult file and queries, e.g.,
    swipl < input.fle
  - You may use ";" to ask "Is there another answer?"
    - The initial query CANNOT have anything on the line after the ";", and
    - There must be a blank line after ";"

SWI-Prolog: Access & Nuance

- SWI-Prolog on Rlogin is located in the directory:
  - /home/staff/arthur/bin/swipl
- swipl prints output to STDERR (file descriptor 2). To redirect output to a file you must precede ">" with a "2":
  - swipl < input.fle 2> output.fle

Prolog – Issues/Limitations

- "Closed World"
  - the only truth is that known to the system
- Efficiency
  - theorem proving can be extremely time consuming
- Resolution order control
  - Prolog always starts with left side of a goal, and always searches database from the top. Have some control by choice of order in the propositions and by structuring database.

Prolog – Issues/Limitations

- Prolog uses backward chaining (start with goal and attempt to find sequence of propositions that leads to facts in the database).
- In some cases forward chaining (start with facts in the database and attempt to find a sequence of propositions that leads to the goal) can be more efficient.
- Prolog always searches depth-first, though breadth-first can work better in some cases.
Prolog - Issues/Limitations

• The Negation Problem -- failure to prove is not equivalent to a logical not
  – not(not(some_goal)) is not necessarily equivalent to some_goal