Chapter 16

Logic Programming

Topics

- Proving Theorems
  - Resolution
  - Instantiation and Unification
- Prolog
  - Terms
  - Clauses
  - Inference Process
  - Backtracking

Predicate Calculus and Proving Theorems

- A use of propositions is to discover new theorems that can be inferred from known axioms and theorems
- *Resolution*: the process of computing inferred propositions from given propositions
  - Resolution principle is similar to the idea of transitivity in algebra.

Resolution

- Making a single inference from a pair of Horn clauses.
  - If h is the head of a Horn clause and it matches with one of the terms of another Horn clause, then that term can be replaced by h.
    - The Horn clauses:

      h ← terms
      t ← t_1, h, t_2
    - The second clause is resolved to

      t ← t_1, terms, t_2

Resolution: example

- Consider the following clauses:

```
speaks(Mary,English).
talkswith(X,Y) ← speaks(X,L), speaks(Y,L), X≠Y
```

- Resolution allow us to deduce:

```
talkswith(Mary,Y) ← speaks(Mary,English),
                    speaks(Y,English), Mary≠Y
```

  - Variables X and L are assigned the values "Mary" and "English" in the second rule.
  - The assignment of values to values during resolution is called *instantiation*.

Unification and Instantiation

- *Unification*: finding values for variables in propositions that allows matching process to succeed
- *Instantiation*: assigning temporary values to variables to allow unification to succeed
- After instantiating a variable with a value, if matching fails, may need to backtrack and instantiate with a different value.

Resolution: Theorem Proving

- Use proof by contradiction.
- *Hypotheses*: a set of pertinent propositions
- *Goal*: negation of theorem stated as a proposition.
- Theorem is proved by finding an inconsistency.

The language Prolog

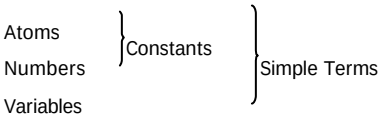
- The most widely used logic programming language.
- Prolog in a nutshell
  - Uses Horn clauses
    - Almost identical notation of Horn clauses, except the implication arrow "←" is replaced by a colon followed by a dash ":-".
  - Implements resolution using strict linear "depth first" strategy and a unification algorithm.

A Prolog Program

- Prolog approach.
  - Describe known facts and relationships.
- A program consists of a sequence of Horn clauses.
  - Clauses are implications of the form
    - p_1 if p_2 … & p_k written as p_1 :- p_2, … ,p_k where every p is a term.
      - p_1 is called the *head* and p_2 , … ,p_k is the *body*.
      - The body is a list of *goals* separated by commas (conjunctions).

Terms

- All Prolog statements are constructed from terms.



Compound/Complex terms (structures)

Terms: atoms

- Atoms are used as names of individuals and predicates.
- Atoms can be constructed in three ways:
  1. Strings of letters, digits and the underscore character, starting with a lower-case letter.
     - `x`
     - `vancouver`
     - `ax123aBCD`
     - `abc_123_etc`

Terms: atoms

2. Strings of special characters .
   - Some of these have a predefined meaning (:-).
   - `-->`
   - `==>`
3. String of characters enclosed in single quotes.
   - It can contain any character.
   - `'Florida'`
   - `'12$12$'`
   - `'  back\\slashes'`
   - `'32'` is an atom, not equal to the number 32.
   - A zero length atom is written ''

Terms: atoms

- Internal representation:
  - Atoms are not character strings, they are locations in a symbol table.
  - Only one copy of each atom is stored.
  - All occurrences are replaced by pointers to its location in the symbol table.
    - `abracadabraabracazam= abracadabraabracazam`
    - `a = b`
    - Comparison takes the same amount of time.
      - Pointers get compared rather than the strings of characters.

Terms: numbers

- Integer
  - Useful for task such as counting the elements of a list.
- Real
  - Not used very much in typical Prolog programming.

Terms: variables

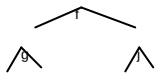
- Stand for objects that we cannot name.
- Begins with a capital letter or '_'.
  - `X`
  - `_value`
  - `Mother`
- A variable can be
  - *Instantiated* when there is an object that it stands for.
  - *Uninstantiated* when what the variable it stands for is not yet known.

Terms: variables

- Question containing a variable:
  - Search through all facts to find an object that the variable could stand for.
- Anonymous variable (_)
  - A special variable that matches anything, but never takes on a value.
  - Successive anonymous variables in the same clause do not take on the same value.

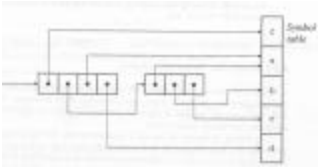
Terms: structures

- Consists of a *functor* followed by a sequence of *arguments*.
  - The functor must be an atom.
  - Arguments can be any kind of terms (including other structures).
  - The outermost functor (`f/2`, in this case) is called the *principal functor* of the structure.
    - `f(g(h,i),j(k,1))`



Terms: structures

- Internal representation
  - A structure is a linked tree made of pointers to its substructures and to entries in the symbol table.



Prolog Clauses

- Prolog clauses are of three types:
  - *Facts:* declare things that are always, unconditionally true.
  - *Rules:* declare things that are true, depending on a given condition.
  - By means of *questions* users can ask the program what things are true.

Prolog Clauses: characteristics

- A Prolog clause:
  - Arbitrary number of arguments (parameters).
  - A predicate that takes `N` arguments is called *N-placed predicate.*
  - A one-place predicate describes a *property* of one individual; a two-place predicate describes a *relation* between two individuals.
  - The number of arguments that a predicate takes is called its *arity*.

Prolog Clauses: characteristics

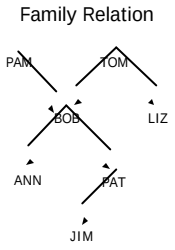
- Two distinct predicates can have the same name if they have different arities.
  - `mother(pam)` meaning Pam is a mother.
  - `mother(pam,bob)` meaning Pam is the mother of Bob.
- A predicate is identified by giving its name, a slash, and its arity.
  - `mother/1.`
  - `mother/2.`
- Every Prolog statement is terminated by a period.

Facts

- Define relationships between objects.
- Sometimes called *ground clauses* because they are the basis from which other information is inferred.
- Facts are clauses that have an empty body.
- Facts are written as:
  `name_of_relationship(object,…,object).`
- A collection of facts and rules is called a *database* or *knowledge base.*

Facts: examples

```
parent(tom,bob).
parent(pam,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).
```



Rules

- Specify things that are true if some condition is satisfied.
- Rules are used to say that a fact depends on a group of other facts.
- A rules consists of a head and a body.
  - Connected by the symbol `':-'` (if).
  - The head describes what fact the rule is intended to define.
  - The body describes the conjunctions of goals that must be satisfied for the head to be true.

Rules: examples

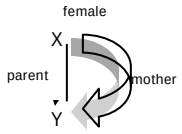
For all X and Y,
Y is an offspring of X if
X is a parent of Y.

```
?-offspring(Y,X):-parent(X,Y).
```

head/conclusion body/condition

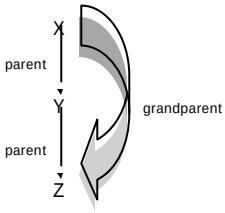
Rules: examples

```
?-mother(X,Y):- parent(X,Y),female(X)
```



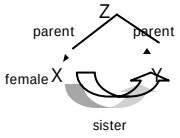
Rules: examples

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



Rules: examples

```
?-sister(X,Y):- parent(Z,X), parent(Z,Y),
                female(X).
```



Rules: exercise

- Define the relation `aunt(X,Y)`

Questions

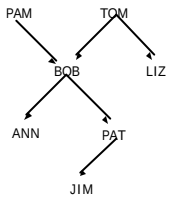
- How do we use Prolog programs?
  - By posing queries (a query is a request for the computer to do something.)
- Questions are clauses that only have the body.
  - Search through the database.
  - Look for facts that *match* the fact in question.
    - Two facts *match* if their predicates and corresponding arguments are the same.
      - YES: Prolog finds a match
      - No: Prolog does not find a match (nothing matches the question vs. false)

Questions: example

User Queries	Prolog's Answers
`?-parent(bob,pat).`	`yes`
`?-parent(liz,pat).`	`no`
`?-parent(tom,ben).`	`no`
`?-parent(X,liz).`	`X=tom`
`?-parent(bob,X).`	`X=ann` `X=pat`

`?-parent(X,Y).`	`X=pam`
	`Y=bob;`
	`X=tom`
	`Y=bob;`
	...

Family Relation



Conjunctions

- More complicated relationships.
- Satisfy two or more separate goals.
- Commas are understood as conjunctions.
- Question containing conjunctions:
  - Try to satisfy each goal in turn (searching the database).
  - All goals have to be satisfied.

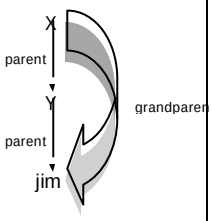
Complex Queries: example

Who is a grandfather of Jim?

1. Who is a parent of Jim?
   Assume that this is some Y.
2. Who is a parent of Y?
   Assume that this is some X.

```
?-parent(X,Y),parent(Y,jim).

X=bob
Y=pat
```



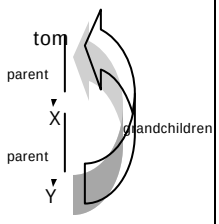
Complex Queries: example

Who are Tom's grandchildren?

1. Who is a child of Tom?
   Assume that this is some X.
2. Who is a child of X?
   Assume that this is some Y.

```
?-parent(tom,X),parent(X,Y).
X=bob
Y=ann

X=bob
Y=pat
```



Complex Queries: example

Do Ann and Pat have a common parent?

1. Who is a parent, X, of Ann?
2. Is (this same) X a parent of Pat?

```
?-parent(X,ann),parent(X,pat).
X=bob
```


Complex Queries: exercises

What will be Prolog's answers to the following questions?

1. `?-parent(jim,X).`
2. `?-parent(X,jim).`
3. `?-parent(pam,X), parent(X,pat).`
4. `?-parent(pam,X),parent(X,Y),parent(Y,jim).`

Formulate in Prolog the following questions:

1. Who is Pat's parent?
2. Does liz have a child?
3. Who is Pat's grandparent?

Matching / Unification

- Two terms (`term`$_1$ and `term`$_2$) can be unified (matched) if they are alike or can be made alike by instantiation.
  - Instantiation: make one variable the same as another.

Matching / Unification

(1) If `term`$_1$ and `term`$_2$ are constants (constants)
- Same *atom* or same *number*.

(2) If `term`$_1$ is a *variable* and `term`$_2$ is any type of term (variables)
- `term`$_1$ is instantiated to `term`$_2$

(3) If `term`$_1$ and `term`$_2$ are *variables* (variables)
- They are instantiated to each other (share values).

(4) If `term`$_1$ and `term`$_2$ are *complex terms* (complex terms)
a. They have the same functor and arity.
b. All their corresponding arguments match.
c. The variable instantiations are compatible.

(5) Two terms match if and only if it follows from the previous four clauses that they match.

Unification: examples

User Queries	Prolog's Answers
`?-2=2.`	`yes`
`?-bob=jim.`	`no`
`?-'bob'=bob.`	`yes`
`?-'2'=2.`	`no`
`?-bob=X.`	`X=bob` `yes`
`?-X=Y.`	`yes`
`?-X=bob, X=jim.`	`no`

`?-kill(shoot(gun),Y))=kill(X,stab(knife)).`	`X= shoot(gun)`
	`Y=stab(knife)`
	`yes`

`?-kill(shoot(gun),stab(knife))=kill(X,stab(Y)).`	`X= shoot(gun)`
	`Y=knife`
	`yes`

Inference Process of Prolog

- Queries are called *goals*
- If a goal is a compound proposition, each of the facts is a *subgoal*
- To prove a goal is true, must find a chain of inference rules and/or facts. For goal Q:

```
B :- A
C :- B
…
Q :- P
```


Inference Process of Prolog

- Process of proving a subgoal is called *matching*, *satisfying*, or *resolution*.
- Bottom-up resolution, forward chaining
  - Begin with facts and rules of database and attempt to find sequence that leads to goal
  - Works well with a large set of possibly correct answers
- Top-down resolution, backward chaining
  - Begin with goal and attempt to find sequence that leads to set of facts in database.

Inference Process of Prolog

  - works well with a small set of possibly correct answers
- Prolog implementations use backward chaining
- When goal has more than one subgoal, can use either
  - Depth-first search: find a complete proof for the first subgoal before working on others
  - Breadth-first search: work on all subgoals in parallel.
- Prolog uses depth-first search

A Simple Prolog Knowledge Base

- A Prolog knowledge base that describes the location of certain North American cities.

```
/* 1 */  located_in(atlanta,georgia).
/* 2 */  located_in(houston,texas).
/* 3 */  located_in(austin,texas).
/* 4 */  located_in(toronto,ontario).
/* 5 */  located_in(X,usa) :- located_in(X,georgia).
/* 6 */  located_in(X,usa) :- located_in(X,texas).
/* 7 */  located_in(X,canada) :- located_in(X,ontario).
/* 8 */  located_in(X,north_america) :-
                                    located_in(X,usa).
/* 9 */  located_in(X,north_america) :-
                                    located_in(X,canada).
```


Inference Process: example

- Query:

```
?- located_in(austin,north_america).
```

Unifies with the head of Clause 8 by instantiating `X` as `austin`.

The right-hand side of Clause 8 becomes the new goal.

Goal: `?- located_in(austin,north_america).`

Clause 8: `located_in(X,north_america) :-`
`located_in(X,usa).`

Instantiation: `X = austin`

New goal: `?-located_in(austin,usa).`

Inference Process: example

Unify the new query with Clause 6:

Goal: `?- located_in(austin,usa).`

Clause 6: `located_in(X,usa) :-`
`located_in(X,texas).`

Instantiation: `X = austin`

New goal: `?-located_in(austin,texas).`

This query matches Clause 3. Since Clause 3 does not contain an "if", no new query is generated and the process terminates successfully.

Backtracking

- If several rules can unify with a query, how does Prolog know which one to use?
- Prolog does not know in advance which clause will succeed but it does know how to black out of blind alleys.
- Prolog tries the rules in order in which they are given in the knowledge base.
  - If a rule does not lead to success, it backs up and tries another

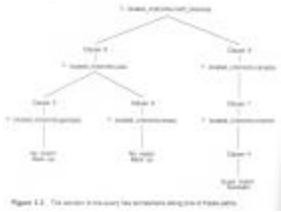
Backtracking

- The query `?-located_in(austin,usa)` will try to unify with Clause 5 and then, when that fails, the computer will back up and try Clause 6.
- A good way to conceive of backtracking is to arrange all possible paths of computation into a tree.
- Consider the query:

```
?- located_in(toronto,north_america).
```


Backtracking: example

- The following tree shows all the paths that the computation might follow.

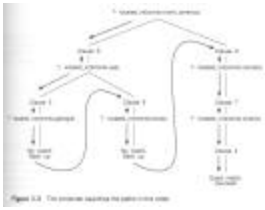


Backtracking: example

- We can prove that Toronto is in North America if we can prove that it is in either the U.S.A. or Canada.
- If we try the U.S.A., we have to try several states and then Canada.
- Almost all paths are blind alleys.
- Only the rightmost one leads to a successful solution.

Backtracking: example

- Same diagram with arrows added to show the order in which the possibilities are tried.



Backtracking: example

- Whenever the computer finds it has gone down a blind alley, it backs up to the most recent query for which there are still untried alternatives, and tries another path.
- When a successful answer is found, the process stops.

Prolog Syntax: comments

- Two ways to delimit comments
  - Anything bracketed by `/*` and `*/`
  - `/* This is a comment */`
- Anything between % and the end of the line
  - `% This is also a comment`

Sicstus

- http://www.cs.sfu.ca/CC/SW/Prolog/
- Linux and SunOS machines
  - CSIL SunOS machines have an additional Prolog implementation: BinProlog.
- Running Sicstus Prolog
  - *orion%* sicstus

Running Sicstus

- Interactive definition mode:
  - The special goal [user] is used to enter interactive definition mode.
  - In interactive mode, Prolog expects goals to establish.
  - ^D (i.e., the ctrl-D key) exits definition mode.
- Consulting a file
  - The special query

    ```
    ?-consult('family.pl').
    ```

    asks prolog to read the definitions from the named file in quotes.
  - Goals must be terminated with a period or Prolog just waits until you enter the period.

Running Sicstus

- To load the same program use `reconsult` instead of `consult`.
  - Otherwise, there will be two copies of it in memory at the same time.
- To exit from Prolog just type the special query `?- halt.`
- If a single query has multiple solutions, Prolog finds one solution and then asks whether to look for another (until all alternatives are found or you stop asking for them).

Running Sicstus

```
?- located_in(X,texas).
       X = houston
       More (y/n) ? y
       X = austin
       More (y/n) ? Y
       no
```

- The *"no"* at the end means there are no more solutions.

- Any of the arguments of a predicate can be queried.
  - `?- located_in(austin,X).` % Names of regions that contain Austin
  - `?- located_in(X,texas).` % Names of cities that are in Texas
  - `?- located_in(X,Y).` % "What is in what?"
  - `?- located_in(X,X).` % "What is in itself?"

Sicstus: examples

- Sample Prolog session

```
1: orion% sicstus
SICStus 3.8.4 (sparc-solaris-5.6): Mon Jun 12 18:49:23 MET DST 2000
Licensed to cs.sfu.ca
| ?- [user].
| member(X, [X|_]).
| member(X, [_|More]) :- member(X, More).
| ^D
|{consulted user in module user, 0 msec 336 bytes}
| ?- ['data.pl'].
{consulting /cs/gard1/dma/family.pl...}
{consulted / cs/gard1/dma/family.pl in module user, 10 msec 160 bytes}
yes
| ?- parent(X,liz).
X = tom;
no
| ?- halt.
2: orion%
```