— Logic programming Lang. (declarative lang.) — ch 16

Symbolic logic as a prog. lang.
predicate calculus

- declarative semantics
- non-procedural programming
  - not describing exactly how a result is to be computed;
  - describing the form of the result.

⇒ sorting algo implementation

- procedural lang. (imperative, functional):
  - details of operations is described in the code.
- vs.
- logic lang.
  - formally describing the characteristics of the result; (sorted list)

⇒ ...

- for each pair,
  - relationship
  - or, for each pair of adjacent eles.

— statements/data — constructed from terms

- constant
  - atom (string)
  - int
- var
  - any string (starts with upper letter)
  - var instantiation at resolution time
  - value/type binding
- structure
  - atomic propositions of predicate calculus
  - format: functor (parameter list)
    - functor: any atom
    - parameter list: any list of atoms, var, structures

↓

↓

∃ 2 basic statement forms

headless
headed
} Horn clause of predicate calculus.

ex) rule statements

ex) single structure
(unconditional assertion (fact))
always True

ex) goal statements

ex) headless Horn clause — only consequence

single structure – fact

ex)
female (shelley).
male (bill).
female (mary).
male (jake).
father (bill, jake).
father (bill, shelley).
mother (mary, jake).
mother (mary, shelley).

bill mary ⇐
jake, shelley

– Rule statement – an ex. of headed Horn clause.

general form:

consequence :- antecedent_expression

consequence: single term; then part; then, this is True
:- : if part; if this is true
antecedent_expression: single term or conjunction; multiple terms with AND or ,

ex) ancestor (mary, shelley) :- mother (mary, shelley)

↓

↓

Using var's

ex) parent(X, Y) :- mother(X, Y).
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
(AND)

X, Y, Z { universal objects, work for any instantiations
ex) (X – mary, Y – shelley), etc.

Operations
ex) X = Y — unify
not(X = Y)

↑ logical propositions are used to describe both known facts and rules that describe logical relationship among facts.

– Goal statements (≡ theorm)
(query)
– system proves (True) or disproves (False)

– syntax – same as headless Horn clause (only consequence).
but, entering { fact/rule statements, goals (queries) } – in separate modes.

– ex).
– { man(fred). – a goal (query)
⇒ system responds Yes or (No) { fact is false or, unable to prove.

ex) – { father(X, mike). (based on the DB of facts/relationships
⇒ system tries to find instantiation of X.

— Inferencing process in Prolog.

— How to prove that a goal (query) is True?

⇒ find a chain of inference rules and/or facts in DB that connects the goal to one or more facts in DB.

eg) goal: Q

⇒ System must find Q as a fact in DB
or, find a fact P_1 and a set of proposition's $P_2 .. P_n$,

s.t. (P_2 :- P_1
P_3 :- P_2
⋮
Q :- P_n) — find these based on matching of terms (inferencing)

eg) man(bob). — a goal (query)

(Case1: if DB contains fact 'man(bob).'
⇒ proof is trivial
Case2: if DB contains
(father(bob). — a fact
man(x) :- father(x). — a rule
⇒ resolution (inferencing) is needed.

— matching a goal to a fact in DB (inferencing, resolution)

vs

way 1. goal ↑ facts/rules — bottom-up resolution (forward chaining)

way 2. goal ↓ facts/rules — top-down resolution (backward chaining)
— good if ∃ small set of candidate answers

prolog

— prolog uses backward chaining

*) goal: man(bob).
propositions in DB:
father(bob).
man(x) :- father(x)

① start from 'man(bob)',
match this goal to the left-side of the rule (2nd propositions)
i.e., man(x) :- father(x)
by instantiating x as 'bob' ⇒ man(bob) :- father(x)

② match the right side of the rule (father(x) ← bob)
with proposition father(bob).

③ return True.

— prolog uses depth-first searching.

finding a complete sequence of propositions (a proof)
for a subgoal, before working on the following subgoals.

ref: breadth-first searching
all subgoals' searching in parallel
→ needs a large amount of memory.

— backtracking — for a compound goal (multiple subgoals)

**). male(x), parent(x, shelley).

subgoal1, subgoal2, subgoal3, ...
True True ↖ is false,
go back to the previous subgoal and try other instantiations

e.g) backtracking

male(x), parent(x, shelley). — a compound goal (query)

1. for subgoal 'male(x)',
   search an instantiation of x s.t. x is male, in DB.
   — assume 'male(mike)' is found in DB.
2. Then, next subgoal 'parent(mike, shelley)' is evaluated.
   — assume fails to prove this (False).
   ⇒ backtrack to subgoal 'male(x)' and search for other instantiation, e.g., 'male(tom)',
   and go ahead to the next subgoal again, ---

repeat this until True or, prove that all males are not parent of shelley.
(False is returned)

Tip: order of subgoals is important

e.g) parent(x, shelley), male(x).

This way is more efficient if ∃ little number of instantiations of x.