

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

vs. procedural lang. (imperative, functional):
 details of operations is described in the code.
logic lang.

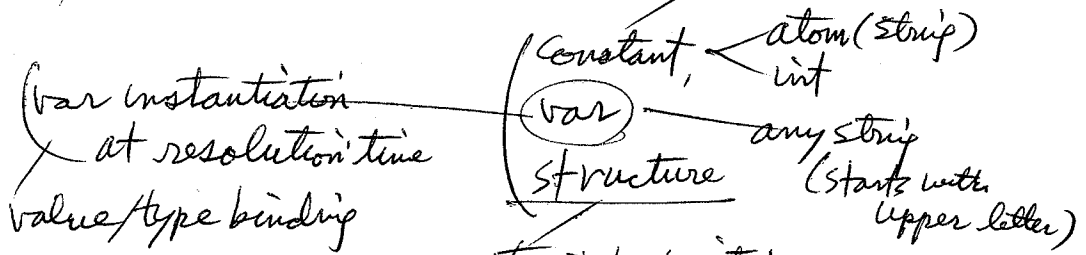
formally describing the characteristics of
 the result.



(Sorted list)

(for each pair,
 relationship
 or, for each pair of adjacent els.)

— statements/data — constructed from terms

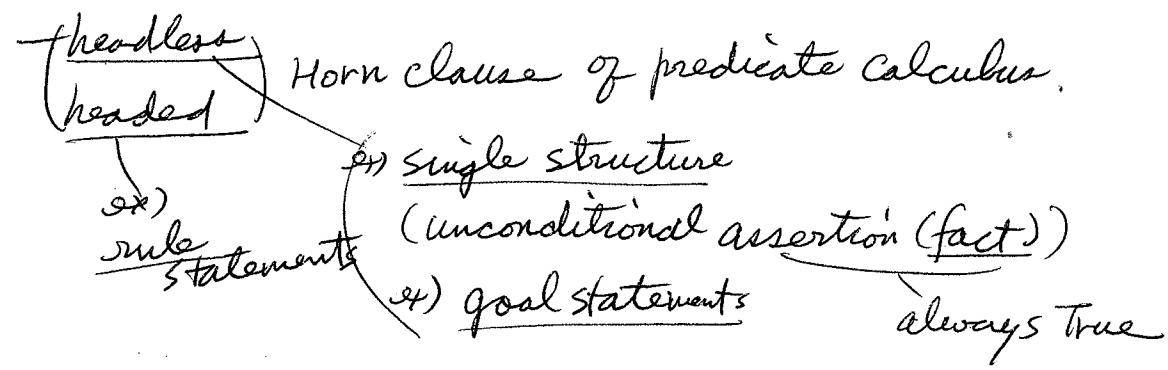


(var instantiation
 at resolution time
 value/type binding)

format: functor (parameter list)
 any atom any list of atoms, var, structures

↓

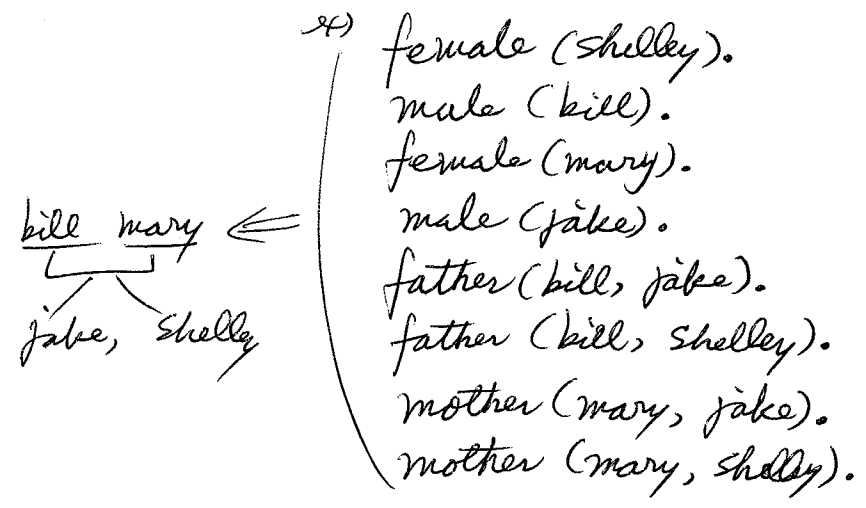
∃ 2 basic statement forms



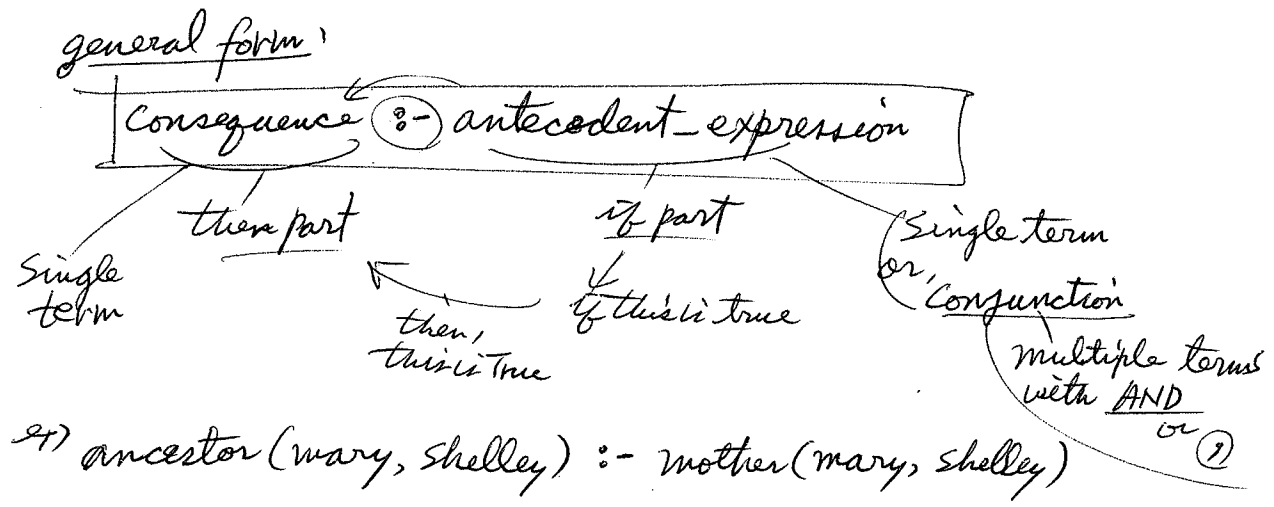
only consequence

ex) headless Horn clause

single structure - fact



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



↓



using vars

ex) $\left(\begin{array}{l} \text{parent}(x, y) :- \text{mother}(x, y). \\ \text{grandparent}(x, z) :- \text{parent}(x, y) \text{ AND } \text{parent}(y, z). \end{array} \right.$

x, y, z { universal objects,
work for any instantiations

ex) (x - mary
y - shelly), etc.

Operations

ex) $\left(\begin{array}{l} x = y \text{ --- infix} \\ \text{not}(x = y) \end{array} \right.$

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

Goal statements (\equiv theorem)

query

— system proves or disproves
(True) (False)

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

— 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.

ex) goal: Q

⇒ system must find Q as a fact in DB

or, find a fact P_1 and a set of propositions $P_2 \dots P_n$,

s.t. $\left(\begin{array}{l} P_2 \leftarrow P_1 \\ P_3 \leftarrow P_2 \\ \vdots \\ Q \leftarrow P_n \end{array} \right)$ find these based on matching of terms (inferencing)

ex) man(bob). — a goal (query)

(Case 1: if DB contains fact 'man(bob).'
⇒ proof is trivial

Case 2: 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)



↓
— prolog uses backward chaining

ex) 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 proposition)

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))
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)

ex). male(x), parent(x, shelly).

subgoal1, subgoal2, subgoal3, ...

True

True

is false,

go back to the previous subgoal and try other

instantiations

24) backtracking

male(x), parent(x, shelly). — 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, shelly)' 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, proving that all males are not parent of shelly.
 (False is returned)

Tip: order of subgoals is important

24) parent(x, shelly), male(x).

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