

A. Prolog differs from FOPC

1. Prolog is more than FOPC: Side effects for predicates: consult/1, halt/0, assertz/1, nl/0, write/1
2. Prolog is less than FOPC:
 - a. As we said, because no NOT or OR in THEN portion
 - a, b:-c. could be replaced by a:-c. b:-c. But a;b :-c. can't be used.
 - b. Also as we said, a:-b. b:-a. will blow the stack instead of assert the logical equivalence of a and b. See pp. 292–293.
 - c. There are no functions in Prolog.

Yes, I know, we represented $\forall x \exists y \text{ loves}(x,y)$ as $\text{loves}(x, f(x))$, with a “function symbol,” but it actually never gets evaluated.

UNIFICATION

- d. = in Prolog does not mean equality, but unification.

See Prolog program `unification.pl` (below)

Do also from text, p. 289:

```
append([], X, Y). append([A|X], Y, [A|Z]) :- append(X, Y, Z).
```

NEGATION

- e. `not` in Prolog does not mean negation but failure to prove from the (domain-specific) axioms. For example the rule `g :- not(american(no_no)).` added to the database on page 2 of this handout. Then `?-g.` returns yes. One can't prove that `no_no` is not an american, but its failure to be proven true is taken to mean false. This is sometimes called the “closed world assumption”: everything not provable is regarded as false.

AS FAILURE

B. Parts to read from Chapter 9, AIMA:

§9.1 all. §9.2 the terms unify, standarize apart, most general unifier (see ?-b. below). §9.3 the terms forward-chaining (and see two Java samples), Horn clauses (from p. 217), writing a constraint satisfaction problem as a statement in FOPC. §9.4 backward chaining, logic programming. §9.5 first three pages, about Skolemization and Skolem functions, mainly as a help to convert logic to Prolog.

C. Sample project for our course.

Demonstrate Otter (pp. 309ff.) or its September successor, Prover9, to the class.

www.cs.unm.edu/~mccune/otter. I saw two easy example to test, at the following links.

Otto: www.cs.unm.edu/~mccune/otter/otter-examples/auto/steam.in

Prover9: www.cs.unm.edu/~mccune/prover9/examples/September-2006/misc/index.html

D. Homework 2, Due Monday, November 6, 2006

This is an individual assignment. Consider p. 268, Exercise 8.6 (a, b, d, e, h, j).

Step 1: Describe in English your ontology for them. For example,

`sells(X, Y, Z)` means that X sells Y to Z.

Be consistent for sentences that are about the same thing.

Step 2: Write each of them in FOPC. I have done Step 2 for (d) in my October 16–18 handout for Chapter 8 in two ways, and you are welcome to use one of them if you'd like.

Step 3: Write Prolog statements for them.

Step 4: Run Prolog on those seven statements treated together as a knowledge base, and test a variety of meaningful queries (goals) against them. Be sure to capture input and output in your report.

Step 5: Pick **one** of the other sentences from Exercise 8.6 and repeat Steps 1 and 2. But now explain why that sentence cannot be represented in Prolog.

Extra credit: Test a query against other sentence using the forward-chaining Java program that is used in `FoldDemo.java` applied to the Skolemized version of the sentence. Does the Java forward-chaining algorithm work, even though the sentence is not a Horn clause?

```
%% unification.pl      Gene Chase, 27 October 2006, version 1.0
%% Some example to illustrate unification in Prolog, which as we saw
%% from lab happens whenever we are goal-seeking, as one step in resolution proof.
```

```
a :- loves(mary, frank) = loves(X,Y).
```

```
b :- thinks(mary, make_happy(X)) =
    thinks(Y,      make_happy(Z)),
    write(X), nl, write(Y), nl, write(Z), nl.
```

```
c :- cause(X,          Y          ) =
    cause(hit(joe, sam), die(sam)),
    write(X), nl, write(Y).
```

```
kill(X, Y) :- cause(X, die(Y)).
```

```
d :- kill(joe, sam) = cause(joe,die(sam)).      %% no!
```

```
e :- cause(X,          die(Y      )) =
    cause(hit(joe, Y), die(frank)),
    write(X), nl, write(Y).
```

```
%% f :- a(joe) = X(joe).          %% not second order logic!
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Java demo of forward chaining can also be done by backward chaining.
% This program agrees with the text, and probably the compiled code for
% Q:\InstructorFiles\Chase_Gene\ai\Code from AIMA Web Site\Java\src\aima\
%   logic\fol\demos\FolDemo.java
% but doesn't agree with the source code of that file. The source code is
% missing missile(m1). See AIMA, page 281
```

```
% Ask ?-criminal(Who).          Answer: Who = west.
%   ?-not(american(no_no)).     Answer: yes.
```

```
% facts
missile(m1).
owns(no_no, m1).
american(west).
enemy(no_no, america).
```

```
% rules
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
sells(west, Missile, no_no) :- missile(Missile), owns(no_no, Missile).
weapon(Missile) :- missile(Missile).
hostile(X) :- enemy(X, Y).
```