nanCS 3500 - Programming Languages and Translators

Prolog:

Pro*grammation en* ***Log****ique* by Alain Colmerauer, Philippe Roussel, Robert Kowalski (1972).

Major Contributions:

- First Logic-based programming language.
- Implementation of a deduction algorithm (Robinson's SLDNF Resolution).

Dialects / Implementations:

- Quintus
- GNU Prolog
- SWI Prolog
- Ciao
- CLP
- ECLiPSe

Comparison Table

	C++	Lisp	Prolog
Paradigm(s)	Procedural, Object Oriented, Generic,	Functional, OOP via CLOS Library	Declarative, Logical. OO via extensions.
Compiled / Interpreted	Compiled	Interpreted & Compiled	Interpreted & Compiled
Main data structures	classes/structs arrays pointers	Atoms, Lists (Common Lisp has arrays, records, etc.)	Atoms, Lists, Functors
Scoping	Static	Static or dynamic	Static
Type bindings	Static	Dynamic	Dynamic
Strongly typed?	Mostly (unions can cause trouble) Many implicit conversions	Yes (but only two types)	Yes
Target applications	Numerical computation, Hardware control,. General application development.	Symbolic Computation. Natural Language Proc. robotics, expert systems, translators, etc.	Symbolic Computation, Natural Language Proc., AI Applications, Expert Systems, Constraint solving.

Notes			

Other Logical Languages:

- Mercury
- Visual Prolog
- CLP(R)
- P#

Introduction:

Calling the Prolog Interpreter (In the cslinux machines):

```
prolog
```

Load a program:

```
['filename'].
```

← *yes, that is a period there…*

Quit the interpreter:

```
?- halt.
```

A simple set of facts:

```
female(lisa).
female(marge).
female(maggie).

male(bart).
male(homer).
male(abe).

parent(marge, lisa).
parent(marge, bart).
parent(marge, maggie).

parent(homer, lisa).
parent(homer, bart).
parent(homer, maggie).

parent(abe, homer).
```

← Lisa is not adopted. see [ref]

Simple Rule Definitions:

```
father(X,Y) :- parent(X,Y),
               male(X).

mother(X,Y) :- parent(X,Y),
               female(X).

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

Examples:

```
?- father(homer, bart).
true .

?- father(X, homer).
X = abe.

?- mother(X, lisa).
X = marge .

?- grandparent(X, bart).
X = abe.
```

Rules with variable comparison:

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

Recursive Rule Definition:

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- ancestor(X,Z),
                 parent(Z,Y).
```

Example:

```
?- ancestor(abe,bart).
true .

?- ancestor(X,maggie).
X = marge ;
X = homer ;
X = abe .
```

List based rules:

```
allfemalelist([]).
allfemalelist([H|T]) :- female(H),
                        allfemalelist(T).
```

.

Other Example:

```
% memb(X,L) => X is a member of L
memb(X,[H|_]) :- X = H.
memb(X,[H|T]) :- memb(X,T).

% rem(L1,X,L2). => L2 is the list that results from removing X from L1.
rem([], X, []).
rem([X|T], X, T).
rem([Y|T], X, [Y|L2]) :- rem(T, X, L2).

% smallest(L,X). => X is the smallest element of L.
smallest([X], X).
smallest([X|T], X) :- smallest(T,Y),
                      X < Y.
smallest([X|T], Y) :- smallest(T,Y),
                      Y < X.

% ssort(L1,L2). => L2 is the sorted version of L1 (selection sort)
ssort([],[]).
ssort(L,[X|Y]) :- smallest(L,X),
                  rem(L,X,L2),
                  ssort(L2,Y).
```

END.