Recursion

• Recursion: define a predicate in terms of *simpler* instances of itself

• Example

  west(R1,R2) ← imm_west(R1,R2).
  west(R1,R2) ← imm_west(R1,R) \land west(R,R2).
  imm_west(r101,r103).
  ...

Recursion

• Idea: *well-founded ordering* among the instances of the relations such that
  – higher level relation is defined in terms of elements in lower levels
  – the lowest level relation is defined by clauses without body.

• Example
  
  west(R1,R2) ← imm_west(R1,R2).
  west(R1,R2) ← imm_west(R1,R) ∧ west(R,R2).
  imm_west(r101,r103).
  ...

Recursion and Mathematical Induction

Recursion
- Top-down (from higher level to lower)
- Simplest level: fact

Mathematical Induction
- Bottom-up (e.g. from n to n+1)
- Base case: n=0
List

• Mathematical abstract concept: ordered sequence of elements

• Abstract data type:
  – Access the head, tail of a nonempty list
  – Construct a list

• Example: Generally written as a list of elements between [ and ]
  – List of students: [ana, bob, cecil, … , zag]
  – List of numbers from 1 to 10: [1, 2, 3, …. , 10]
Representation

- A list can be defined in terms of two relations:
  - \( elt(L, K, E) \) is true if \( E \) is the \( k \)th element of list \( L \)
  - \( size(L, N) \) is true if list \( L \) has \( N \) elements

- This means that a list can be specified by a KB whose logical consequences are atoms of the forms \( elt(L, K, E) \) and \( size(L, N) \).
Example

List of integer between 1 and 10

\[
\begin{align*}
\text{elt(Int1\_10, 1, 1)} & \leftarrow \text{elt(Int1\_10, 6, 6)} \\
\text{elt(Int1\_10, 2, 2)} & \leftarrow \text{elt(Int1\_10, 7, 7)} \\
\text{elt(Int1\_10, 3, 3)} & \leftarrow \text{elt(Int1\_10, 8, 8)} \\
\text{elt(Int1\_10, 4, 4)} & \leftarrow \text{elt(Int1\_10, 9, 9)} \\
\text{elt(Int1\_10, 5, 5)} & \leftarrow \text{elt(Int1\_10, 10, 10)} \\
\text{size(Int1\_10, 10)} & \leftarrow \\
\end{align*}
\]

Note: The \textit{kth} element counted from 1 (Could be counted from 0).
Other representation

• A list is
  – either an empty list 
  – or a head (an element) followed by a list, called as tail.

• We can represent as
  – either a constant nil;
  – or a term of the from cons(H,T) where cons is a function symbol, H represents the head and T represents the tail.
Example
List of integer between 1 and 10
cons(1, cons(2, cons(3, cons(4, cons(5, cons(6, cons(7, cons(8, cons(9, cons(10, nil)
))))))
)))))
))))
Checking if L is a list

\[
\begin{align*}
\text{list}([]) & \leftarrow \\
\text{list}(\text{cons}(H,T)) & \leftarrow \text{list}(L).
\end{align*}
\]
Query

append(nil, X, X) ←
append(cons(H,T), Y, cons(H, Z)) ← append(T, Y, Z).

? append(cons(a,cons(b,nil)), cons(c, cons(d, nil)), Z).
Answer: Z = cons(a,cons(b,cons(c, cons(d, nil)))))
How?

? append(A, B, cons(a,cons(b,cons(c,cons(d, nil))))).
Five answers.
How?
Append

append(X, Y, Z) is true if Z is a list beginning with the elements of X followed by the elements of Y

append(nil, X, X) ←
append(cons(H,T), Y, cons(H, Z)) ←
      append(T, Y, Z).
Notation

- *nil* is written as []
- *cons*(H, T) is written as [H|T]
- Short hand: [P|[Q]] as [P,Q]
  - [a|[d|nil]] is [a,d|nil]
  - [a,b,c|[d,e,f]] is [a,b,c,d,e,f]
- Rewritten *append*

append([], X, X) ←
append([H|T], Y, [H|Z]) ← append(T, Y, Z).
Useful Operations on Lists

- Membership function: $\text{member}(X,L)$ is true if $X$ is an element of $L$.
- Size of a list: $\text{length}(L,N)$ is true if $N$ is the number of elements of $L$.
- Reversing a list: $\text{reverse}(L,R)$ is true if $R$ is the reverse of $L$.
- Difference of two lists: $\text{diff}(L1,L2,R)$ is true if $R$ is the list obtained from $L1$ by removing from $L1$ all elements of $L2$. 
Using Prolog

- Login to Solaris/Linux machine
- Prolog interpreter: sicstus
- compile a file named test.pl: compile(‘test.pl’).
- Prolog program:
  - %: comments
  - Commands end with ‘.’
  - ‘:-’ is used instead of ‘←’
  - ‘,’ is used instead of ‘∧’
The ‘append.pl’ program

% this is my append.pl program
% append(X,Y,Z) is true iff Z is a list consisting
% of elements of X followed by elements of Y
append(nil, X, X).
append(cons(H,T), Y, cons(H, Z)):- append(T, Y, Z).
% added checking for being a list
list(nil).
list(cons(H,T)):- list(T).
Asking queries

:- append(cons(a,cons(b,nil)),cons(c,cons(d,nil)), X).
we get
X = cons(a,cons(b,cons(c,cons(d,nil)))) ?
typing ;
no
:- append(cons(a,cons(b,nil)),cons(c,cons(d,nil)),cons(a,nil)).
we get
no
:- append(A, B, cons(a,cons(b,cons(c,cons(d,nil)))))
we get A = nil, B = cons(a,cons(b,cons(c,cons(d,nil)))))? ;
we get A = cons(a,nil), B = cons(b,cons(c,cons(d,nil))))? ;
...

The ‘west.pl’ program

% Computational Intelligence: a logical approach.
% Prolog Code. Figure 2.3 & Example 2.13.
% Copyright (c) 1998, Poole, Mackworth, Goebel and
Oxford University Press

% imm_west(R1,R2) is true if room R1 is immediately
west of room R2

imm_west(r101,r103).
imm_west(r103,r105).
imm_west(r105,r107).
imm_west(r107,r109).
imm_west(r109,r111).
imm_west(r131,r129).
imm_west(r129,r127).
imm_west(r127,r125).

% imm_east(R1,R2) is true if room R1 is immediately
east of room R2

imm_east(r101,r103).
imm_east(r103,r105).
imm_east(r105,r107).
imm_east(r107,r109).
imm_east(r109,r111).
imm_east(r131,r129).
imm_east(r129,r127).
imm_east(r127,r125).

% next_door(R1,R2) is true if room R1 is next door to
room R2

next_door(R1,R2) :-
  imm_east(R1,R2).
next_door(R1,R2) :-
  imm_west(R1,R2).

% two_doors_east(R1,R2) is true if room R1 is two doors
east of room R2

two_doors_east(R1,R2) :-
  imm_east(R1,R),
  imm_east(R,R2).

% west(R1,R2) is true if room R1 is somewhere west of
room R2

west(R1,R2) :-
  imm_west(R1,R2).
wR1,R2) :-
  imm_west(R1,R),
  west(R,R2).
% course(C) is true if C is a university course
course(312).
course(322).
course(315).
course(371).

department(312,comp_science).
department(322,comp_science).
department(315,math).
department(371,physics).

% student(S) is true if S is a student
student(mary).
student(jane).
student(john).
student(harold).

% female(P) is true if person P is female
female(mary).
female(jane).

% enrolled(S,C) is true if student S is enrolled in
%     course C
enrolled(mary,322).
enrolled(mary,312).
enrolled(john,322).
enrolled(mary,315).
enrolled(jane,312).
enrolled(jane,322).
enrolled(john,315).
enrolled(harold,322).
enrolled(mary,315).
enrolled(jane,312).
enrolled(jane,322).

% cs_course(C) is true if course C is offered in
%     comp_science
% math_course(C) is true if course C is offered in ..
%     math

% cs_or_math_course(C) is true if course C is
%     offered in CS or math

cs_course(C) :- department(C,comp_science).
math_course(C) :- department(C,math).

% cs_or_math_course(C) is true if course C is
%     offered in CS or math

% in_dept(S,D) is true if student S is enrolled
%     in a course offered in department D

in_dept(S,D) :- enrolled(S,C), department(C,D).

% example query
% ? enrolled(S,C), department(C,D).