CMSC 330: Organization of Programming Languages

Logic Programming with Prolog Lists
Review: Execution = Search

- Prolog execution: Goal-directed search
  - Query = predicate you wish to prove is true

- Key feature: unification
  - Two terms unify if they are identical, or they can be made identical by substituting variables
    - is_bigger(X, gnat) = is_bigger(horse, gnat) when X=horse
    - execution goal is often to discover such X

- Attempt to unify goal with head of a rule
  - If succeeds, clauses in body become subgoals
  - Continue until all subgoals satisfied
    - If search fails, backtrack and try untried subgoals
Review: Equality

Not all forms of equality are the same!

• \( p = q \) iff \( p \) unifies with \( q \)

• \( p \) is \( q \) iff \( p \) unifies with \( q' \) where \( q' \) is \( q \) evaluated
  - Meaning that \( q' \) is treated as an arithmetic expression, and run as such

• \( p =:= \) iff \( p' \) unifies with \( q' \) where \( q' \) is \( q \) evaluated and \( p' \) is \( p \) evaluated

• \( p == q \) iff \( p \) and \( q \) are identical
  - No substitutions or evaluations permitted
Warmup: What is the query result?

\[ \text{john}(C, E, N, A) : - \]
\[ C = N, \]
\[ E = A, \]
\[ C = 2 + 3. \]

? - \text{john}(5, 1, 5, 1).

A. true
B. false
Warmup: What is the query result?

john(C, E, N, A) :-
  C = N,
  E = A,
  C = 2 + 3.
?- john(5, 1, 5, 1).

A. true
B. false. = does not evaluate 2+3
Lists In Prolog

- \([a, b, 1, ‘hi’, [X, 2]]\)
- But really represented as compound terms
  - \([\ ]\) is an atom
  - \([a, b, c]\) is represented as \(.(a, .(b, .(c, []))))\)
- Matching over lists
  \[- [X, 1, Z] = [a, _, 17]\]
  \- X = a,
  \- Z = 17.
List Deconstruction

- Syntactically similar to Ocaml: [H|T] like h::t
  
  \(\text{?- [Head} \mid \text{Tail]} = [a, b, c].\)
  
  Head = a,
  Tail = [b, c].

  \(\text{?- [1,2,3,4]} = [_], X \mid _].\)
  
  X = 2

- This is sufficient for defining complex predicates

- Let’s define \text{concat}(L1, L2, C)
  
  \(\text{?- concat([a,b,c], [d,e,f], X).}\)
  
  X = [a,b,c,d,e,f].\)
Example: Concatenating Lists

To program this, we define the “rules” of concatenation

- If L1 is empty, then C = L2

\[
\text{concat( [ ], L2, L2 ).}
\]

- Prepending a new element to L1 prepends it to C, so long as C is the concatenation of L1 with some L2

\[
\text{concat( [E | L1], L2, [E | C] ) :-}
\text{concat(L1, L2, C).}
\]

... and we’re done
Why Is The Return Value An Argument?

- Now we can ask what inputs lead to an output.

?- concat(X, Y, [a,b,c]).

\[
\begin{align*}
X &= [], \\
Y &= [a, b, c] ; \\
X &= [a], \\
Y &= [b, c] ; \\
X &= [a, b], \\
Y &= [c] ; \\
X &= [a, b, c], \\
Y &= [ ] ;
\end{align*}
\]

User types ; to request additional answers.
Quiz 1: T/F: This is a Valid Prolog List

[3, 4, 'papaya', blueberry]

A. True  
B. False
Quiz 1: T/F: This is a Valid Prolog List

[3, 4, 'papaya', blueberry]

A. True
B. False
Quiz 2: What does this query return?

?- [a|T] = [a,b,c,[d,a],[1,2],list].

A. T = [b, c, [d, a], [1, 2], list].
B. false
C. T = [d, a]
D. T = list
Quiz 2: What does this query return?

?- [a|T] = [a,b,c,[d,a],[1,2],list].

A. T = [b, c, [d, a], [1, 2], list].
B. false
C. T = [d, a]
D. T = list
Quiz 3: What does mystery(A,L) do?

mystery(X, [H|T]) :- X = H.
mystery(X, [H|T]) :- mystery(X,T).

A. Evaluates to false if A is contained in list L
B. Evaluates to true if A is contained in list L
C. Assigns the last element in L to A
D. Assigns the first element in L to A
Quiz 3: What does mystery(A,L) do?

mystery(X, [H|T]) :- X = H.
mystery(X, [H|T]) :- mystery(X,T).

A. Evaluates to false if A is contained in list L
B. Evaluates to true if A is contained in list L
C. Assigns the last element in L to A
D. Assigns the first element in L to A
Quiz 4: What’s result of mystery(A,B) ?

mystery(L1,L2) :-
  L1 = [H|T1],
  L2 = [H,H|T2].

A. true if A and B have equal lengths
B. true if the first element in A is equal to the first and the last element in B.
C. true if the first element in A is equal to the first and the second element in B.
D. true if the first element in A is equal to the last element in B.
Quiz 4: What’s result of mystery(A,B) ?

mystery(L1,L2) :-
   L1 = [H|T1],
   L2 = [H,H|T2].

A. true if A and B have equal lengths
B. true if the first element in A is equal to the first and the last element in B.
C. true if the first element in A is equal to the first and the second element in B.
D. true if the first element in A is equal to the last element in B.
Built-in List Predicates

- **length(List,Length)**
  
  ```
  ?- length([a, b, [1,2,3] ], Length).
  Length = 3.
  ```

- **member(Elem,List)**
  
  ```
  ?- member(duey, [huey, duey, luey]).
  true.
  
  ?- member(X, [huey, duey, luey]).
  X = huey; X = duey; X = luey.
  ```

- **append(List1,List2,Result)**
  
  ```
  ?- append([duey], [huey, duey, luey], X).
  X = [duey, huey, duey, luey].
  ```
Built-in Predicates

- sort(List, SortedList)
  
  `- sort([2, 1, 3], R).
  R = [1, 2, 3].

- findall(Elem, Predicate, ResultList)
  
  `- findall(E, member(E, [huey, duey, luey]), R).
  R = [huey, duey, luey].

- setof(Elem, Predicate, ResultSortedList)
  
  `- setof(E, member(E, [huey, duey, luey]), R).
  R = [duey, huey, luey].

- See documentation for more
  
Example – Towers of Hanoi

- Problem
  - Move stack of disks between pegs
  - Can only move top disk in stack
  - Only allowed to place disk on top of larger disk
Example – Towers of Hanoi

- To move a stack of $n$ disks from peg $X$ to $Y$
  - Base case
    - If $n = 1$, move disk from $X$ to $Y$
  - Recursive step
    1. Move top $n-1$ disks from $X$ to 3$^{rd}$ peg (Z)
    2. Move bottom disk from $X$ to $Y$
    3. Move top $n-1$ disks from 3$^{rd}$ peg (Z) to $Y$

Iterative algorithm would take much longer to describe!
Towers of Hanoi

Code

move(1,X,Y,_) :-
    write('Move top disk from '), write(X),
    write(' to '), write(Y), nl.
move(N,X,Y,Z) :-
    N>1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).
Prolog Terminology

- A query, goal, or term where variables do not occur is called ground; else it’s nonground
  - foo(a,b) is ground; bar(X) is nonground

- A substitution $\theta$ is a partial map from variables to terms where $\text{domain}(\theta) \cap \text{range}(\theta) = \emptyset$
  - Variables are terms, so a substitution can map variables to other variables, but not to themselves

- A is an instance of B if there is a substitution such that $A = B\theta$  
  ![The substitution $\theta$ applied to B](image)

- C is a common instance of A and B if it is an instance of A and an instance of B
Prolog’s Algorithm Solve()

Solve(goal \(G\), program \(P\), substitution \(\theta\)) =

- Suppose \(G\) is \(A_1,\ldots,A_n\). Choose goal \(A_1\).
- For each clause \(A : - B_1,B_2,\ldots,B_k\) in \(P\),
  - if \(\theta_1\) is the mgu of \(A\) and \(A_1\theta\) then
    - If Solve\(\{B_1,\ldots,B_k,A_2,\ldots,A_n\}, P, \theta \cdot \theta_1\) = some \(\theta'\) then return \(\theta'\)
    - (else it has failed, so we continue the for loop)
  - (else unification has failed, so try another rule)
- If loop exits return fail
- Output: \(\theta\) s.t. \(G\theta\) can be deduced from \(P\), or fail