Example programs

York University CSE 3401
Vida Movahedi
Overview

• Classifying terms
  – Weight Conversion

• Working with Lists

• Working with Structures
  – Board Example

• Linked Lists

• Binary Trees

[ref.: Clocksin, Chap 6 & 7 ]
[also Prof. Zbigniew Stachniak’s notes]
Weight conversion

• Problem:
  – Convert Pounds to Kilos and vice versa
  – Show an error message and fail if no inputs given
  – Show an error message and fail if given input is not a number

Some useful built-in predicates:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var(X)</td>
<td>succeeds if X is a variable and is not instantiated</td>
</tr>
<tr>
<td>nonvar(X)</td>
<td>succeeds if var(X) fails</td>
</tr>
<tr>
<td>atom(X)</td>
<td>succeeds if X stands for an atom e.g. adam, ‘George’, ...</td>
</tr>
<tr>
<td>number(X)</td>
<td>succeeds if X stands for a number</td>
</tr>
<tr>
<td>atomic(X)</td>
<td>succeeds if X stands for a number or an atom</td>
</tr>
<tr>
<td>integer(X)</td>
<td>Succeeds if X stands for an integer.</td>
</tr>
</tbody>
</table>
Weight Conversion- code

\[
\text{convert}(	ext{Pounds}, \text{Kilos}) :- \\
\text{\% If no inputs given} \\
\text{var(Pounds), var(Kilos), !,} \\
\text{write('No inputs!'), nl, fail.}
\]

\[
\text{convert}(	ext{Pounds}, \_):- \\
\text{\% If Pounds is known, but not a number} \\
\text{nonvar(Pounds), \+number(Pounds), !,} \\
\text{write('Inputs must be numbers!'), nl, fail.}
\]

\[
\text{convert}(\_, \text{Kilos}) :- \\
\text{\% If Kilos is known, but not a number} \\
\text{nonvar(Kilos), \+number(Kilos), !,} \\
\text{write('Inputs must be numbers!'), nl, fail.}
\]

\[
\text{convert}(	ext{Pounds}, \text{Kilos}) :- \\
\text{\% If Pounds is known} \\
\text{number(Pounds), !,} \\
\text{Kilos is Pounds * 0.45359.}
\]

\[
\text{convert}(	ext{Pounds}, \text{Kilos}) :- \\
\text{\% Otherwise} \\
\text{Pounds is Kilos/0.45359.}
\]
17 ?- convert(X,Y).  
No inputs!  
false.

18 ?- convert(20,Y).  
Y = 9.0718.

19 ?- convert(X,9).  
X = 19.8417.

20 ?- convert(20,9.0718).  
true.

21 ?- X=5, convert(X,Y).  
X = 5,  
Y = 2.26795.

22 ?- convert(X,a).  
Inputs must be numbers!  
false.
Working with Lists

• Find the first element of a list.
  \[ \text{first}(X, [X|\_]). \]

• Find the last element of a list.
  \[ \text{last}(X, [X]). \]
  \[ \text{last}(X, [H|T]) :- \text{last}(X, T). \]

• Shift the elements of a list to left.
  \[ \text{lshift}([H|T], L) :- \text{append}(T, [H], L). \]

?- lshift([1, 2, 3, 4, 5], L).
L = [2, 3, 4, 5, 1].
Working with Lists (2)

?- lshift(L, [1, 2, 3, 4, 5]).
L = [5, 1, 2, 3, 4].

• Shift the elements of a list to the right.

   rshift(L, R):- lshift(R, L).

• Shift the elements of a list to the right N times.

   good(N):- integer(N), N >= 0.
   rshift(L, N, R):-
   \+good(N), !,
   write('N must be a known positive integer.'), nl, fail.
   rshift(L, 0, L).
   rshift(L, N, R):-
   N>0, rshift(L, R1), N1 is N-1, rshift(R1, N1, R).
Working with Lists (3)

• Change the $N^{th}$ element of a list
  – Assuming we already checked for the possible errors (e.g. $N<1$ or $N>\text{length of list}$)
  – $\text{setPosition}(L_1, N, X, L_2)$ returns list $L_2$ which is the same as list $L_1$, except that its $N^{th}$ element is changed to $X$.

?- setPosition([1, 2, 3, 4], 2, z, L).
L = [1, z, 3, 4]

setPosition([_|L], 1, X, [X|L]).
setPosition([H|L1], N, X, [H|L2]):-
  N > 1, N1 is N-1,
  setPosition(L1, N1, X, L2).

See more examples of list processing in Clocksin, Section 7.5.
Board example:
Input a board position number

• Get an **integer** from 1 to 9 from user, set the corresponding board position to ‘x’.

  ```prolog
  getXPosition(N):-
      write('Enter a position (1-9): '),
      read(N),
      integer(N), N > 0, N < 10, !.
  ```

  ```prolog
  getXPosition(N):= getXPosition(N).
  ```

• or use **`repeat`**

  ```prolog
  getXPosition(N):-
      repeat,
      write('Enter a position (1-9): '),
      read(N),
      integer(N), N > 0, N < 10, !.
  ```
Working with structures

• The board is a structure $b(B_1, B_2, ..., B_9)$
  For example, this board is shown as
  $b(e, x, o, e, x, e, e, e)$

• How can we access the components, especially if we don’t know anything about the structure.

• Useful built-in predicates:

<table>
<thead>
<tr>
<th>$functor(T, F, N)$</th>
<th>means “$T$ is a structure with functor $F$ and arity $N$”.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$arg(N, T, A)$</td>
<td>Returns the $N^{th}$ argument of $T$.</td>
</tr>
<tr>
<td>$T =.. L$</td>
<td>means “$L$ is a list of functor of $T$ and its arguments”</td>
</tr>
</tbody>
</table>
Working with structures - examples

?- functor(s(a,b,c), F, N).
F = s, N = 3.
?- functor(c, F, N).
F = c, N = 0.
?- functor(T, book, 2).

?- arg(2, s(a,b,c), X).
X = b.
?- arg(2, [a, b, c], X).
X = [b, c].

?- s(a,b,c) =.. L.
L = [s, a, b, c].
?- s(a,b,c) =.. [H|L].
H = s, L = [a, b, c].
?- T =.. [g,1].
T = g(1).

?- [a, b, c] =.. [H|T].
H = ‘.’, T = [a, [b,c]].
Board example:
Set a board position

• Set a board position to X

  \texttt{setPosBoard}(\texttt{OldB}, \texttt{N}, \texttt{X}, \texttt{NewB}) :-
  \texttt{OldB} =.. \texttt{[H|L1]},
  \texttt{setPosition(L1, N, X, L2)},
  \texttt{NewB} =.. \texttt{[H|L2]}.

• Ask the position from user and set that position to ‘x’

  \texttt{nextX}(\texttt{OldB, NewB}) :-
  \texttt{getXPosition(N)},
  \texttt{setPosBoard(OldB,N,x,NewB)}. 
Board example:
Set a board position if available

But we also have to make sure the board position is available

\[
\text{nextX}(\text{OldB}, \text{NewB}):-
\]
\[
\quad \text{getXPosition}(N), \\
\quad \text{checkPosition}(\text{OldB}, N),!, \\
\quad \text{setPosBoard}(\text{OldB}, N, x, \text{NewB}).
\]

\[
\%	ext{ ask where to play}
\%	ext{ is it available}
\%	ext{ set the board to x}
\%	ext{ else error message}
\]

\[
\text{nextX}(\text{OldB}, \text{NewB}):-
\quad \text{write('Not an empty board position!'),} \\
\quad \text{nl}, \\
\quad \text{nextX}(\text{OldB}, \text{NewB}).
\]

\[
\text{nextX}(\text{OldB}, \text{NewB}):-
\]

Board example:
Checking a position on board

• Is the board position containing an ‘e’?
  – Reminder: we decided to have e in any empty position.

checkPosition(B, N) :-
  arg(N, B, X),    % get position N on board
  X = e.          % check if it is e
Linked Lists

• We can define linked lists as a structure with two arguments: data and link

\[ \text{llist}(\text{Data}, \text{Link}) \]

\[ \text{llist}(34, \text{llist}(31, \ldots, \text{llist}(2, \text{llist}(69, \text{end})) \ldots)) \]

– Used the constant ‘end’ to mark the end of the linked list.
– It is possible to have a more complicated Data, or more arguments for llist.
Linked Lists - search & insert

• Write \( \text{search}(X, LL) \) which succeeds if \( X \) is a data in a linked list \( LL \).

\[
\begin{align*}
\text{search}(\_, \text{end}) & :\text{ fail}. \\
\text{search}(X, \text{llist}(X, \_)) & . \\
\text{search}(X, \text{llist}(\_,\text{Rest})) & :\text{ search}(X, \text{Rest}).
\end{align*}
\]

• Write \( \text{insert}(X, LL1, LL2) \) which inserts \( X \) in front of \( LL1 \) to get \( LL2 \).

\[
\text{insert}(X, LL1, \text{llist}(X, LL1)).
\]

• Exercise:
  – write \( \text{delete}(X, LL1, LL2) \) which deletes all occurrence of \( X \) in \( LL1 \) to get \( LL2 \).
Ordered Linked Lists

- Write $\text{add}(X, LL1, LL2)$ which inserts $X$ in an **ordered** link list $LL1$ to get $LL2$.

  $\text{add}(X, \text{end}, \text{llist}(X, \text{end})).$
  $\text{add}(X, \text{llist}(Y, \text{Rest}), \text{llist}(X, \text{llist}(Y, \text{Rest}))) :- X =< Y.$
  $\text{add}(X, \text{llist}(Y, \text{Rest}), \text{llist}(Y, \text{Rest2})) :- X > Y,$
  $\text{add}(X, \text{Rest}, \text{Rest2}).$

?- $\text{add}(34, \text{end}, R1), \text{add}(31, R1, R2), \text{add}(2, R2, R3), \text{add}(69, R3, \text{Result}).$
  $\text{R1} = \text{llist}(34, \text{end}),$
  $\text{R2} = \text{llist}(31, \text{llist}(34, \text{end})),$
  $\text{R3} = \text{llist}(2, \text{llist}(31, \text{llist}(34, \text{end})),$
  $\text{Result} = \text{llist}(2, \text{llist}(31, \text{llist}(34, \text{llist}(69, \text{end}))));$
  false
Ordered Linked Lists (cont.)

• Exercise:
  1. Modify add to skip adding an element if it is already in the linked list.
  2. Modify add to work with terms, use $X@<Y, X@Y$, etc.

• Write $\text{del}(X, L1, L2)$ to delete $X$ from an ordered linked list:
  $\text{del}(X, \text{Old}, \text{New}) :\text{-} \text{add}(X, \text{New}, \text{Old})$.

?- add(5, llist(1, llist(2, end)), R),
   del(2, R, Final).
R = llist(1, llist(2, llist(5, end))),
Final = llist(1, llist(5, end)) ;
false.
Binary Trees

• Each node **Root** in a binary tree has two children, **Left** and **Right**.
  
  \[ t(\text{Left}, \text{Root}, \text{Right}) \]

• Unless it is a leaf, which can be denoted as ‘end’.
  
  \[ t(t(t(\text{end}, 6, \text{end}), \quad 1, t(t(\text{end}, 2, \text{end})), \quad 5, t(\text{end}, 6, \text{end}))) \]
Binary Trees- counting elements

• Counting the elements in a binary tree

\[
\text{count}(\text{end}, 0).
\]

\[
\text{count}(t(\text{Left}, \text{Root}, \text{Right}), N) :-
\]

\[
\begin{align*}
\text{count}(\text{Left}, N1), \\
\text{count}(\text{Right}, N2), \\
N & \text{ is } N1 + N2 + 1.
\end{align*}
\]

?- \text{count}(t(t(t(\text{end}, 6, \text{end}), 1, t(\text{end}, 2, \text{end})), 5, t(\text{end}, 6, \text{end})), N).

\[N = 5.\]
Sorted Binary Trees

• A binary tree is sorted if
  \[ \text{Left} < \text{Root} \leq \text{Right} \]

• Searching for an item in a sorted binary tree:

  \[
  \text{lookup}(\text{Item}, t(\text{Left}, \text{Item}, \text{Right})).
  \]
  \[
  \text{lookup}(\text{Item}, t(\text{Left}, \text{Root}, \text{Right})):\]
  \[
  \text{Item} < \text{Root},
  \text{lookup}(\text{Item}, \text{Left}).
  \]
  \[
  \text{lookup}(\text{Item}, t(\text{Left}, \text{Root}, \text{Right})):\]
  \[
  \text{Item} > \text{Root},
  \text{lookup}(\text{Item}, \text{Right}).
  \]
Sorted Binary Trees - add items

- Adding an item in a sorted binary tree

\[
\text{addT}(X, \text{end}, t(\text{end}, X, \text{end})). \quad \% \text{ if empty tree}
\]

\[
\text{addT}(X, t(L, \text{Root}, R), t(L1, \text{Root}, R)):-
\]

\[
X < \text{Root}, \quad \text{addT}(X, L, L1).
\]

\[
\text{addT}(X, t(L, \text{Root}, R), t(L, \text{Root}, R1)):-
\]

\[
X \geq \text{Root}, \quad \text{addT}(X, R, R1).
\]

?- addT(3,t( t( t(\text{end}, 6,\text{end}),1, t(\text{end},2,\text{end})), 5, t(\text{end},6,\text{end})), L).
L = t(t(t(\text{end}, 6, \text{end}), 1, t(\text{end}, 2, t(\text{end}, 3, \text{end}))), 5, t(\text{end}, 6, \text{end}))

Note this is not a sorted tree, just using the previous example
Sorted Binary Trees - add items

After addT(3, ..)

Note this is not a sorted tree, just using the previous example
Sorted Binary Trees - delete items

- Deleting an item from a sorted binary tree

\[
\text{delT}(X, t(\text{end}, X, R), R).
\]
\[
\text{delT}(X, t(L, X, \text{end}), L).
\]
\[
\text{delT}(X, t(L, X, R), t(L, Y, R1)):- \quad \text{delMin}(R, Y, R1).
\]
\[
\text{delT}(X, t(L, A, R), t(L1, A, R)):- \quad X < A, \\
\quad \text{delT}(X, L, L1).
\]
\[
\text{delT}(X, t(L, A, R), t(L, A, R1)):- \quad X > A, \\
\quad \text{delT}(X, R, R1).
\]
\[
\text{delMin}(t(\text{end}, Y, R), Y, R).
\]
\[
\text{delMin}(t(L, \text{Root}, R), Y, t(L1, \text{Root}, R)):- \quad \text{delMin}(L, Y, L1).
\]

- Exercise: What is the property of node Y in \( \text{delMin}(T1, Y, T2) \)?