Example programs

York University
Department of Computer Science and Engineering
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

convert(Pounds, Kilos):- % If no inputs given
    var(Pounds), var(Kilos), !,
    write('No inputs!'), nl, fail.

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

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

convert(Pounds, Kilos):- % If Pounds is known
    number(Pounds), !,
    Kilos is Pounds * 0.45359.

convert(Pounds, Kilos):- % Otherwise
    Pounds is Kilos/0.45359.
### Weight conversion- queries

?- convert(X,Y).
No inputs!
false.

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

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

?- convert(20,9.0718).
true.

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

?- 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.
  \[ \text{lshift}([H|T], \ L) \] :- \[ \text{append}(T, \ [H], \ L) \].
  \[ ?- \text{lshift}([1, 2, 3, 4, 5], \ L) \].
  \[ L = [2, 3, 4, 5, 1] \].
Working with Lists (2)

• Note:

?- 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).
• Change the N\textsuperscript{th} element of a list
  – Assuming we already checked for the possible errors (e.g. N<1 or N> length of list)
  – \textit{setPosition} \((L1, N, X, L2)\) returns list L2 which is the same as list L1, except that its N\textsuperscript{th} element is changed to X.

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

\[
\text{setPosition([-|L], 1, X, [X|L]).}
\]

\[
\text{setPosition([H|L1], N, X, [H|L2]):-}
\]
N > 1,
N1 is N-1,
\text{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’.

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

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

• or use repeat

  `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 \( b(B_1, B_2, \ldots, B_9) \)
  For example, this board is shown as
  \( b(e, x, o, e, x, e, e, e, e) \)

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

• Useful built-in predicates:

| \( \text{functor}(T, F, N) \) | means “\( T \) is a structure with functor \( F \) and arity \( N \)”.
| \( \text{arg}(N, T, A) \) | Returns/ matches the \( N^{th} \) argument of \( T \) in/ with \( A \).
| \( T =.. L \) | means “\( L \) is a list of functor of \( T \) and its arguments” |
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

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

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

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

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

\[
\text{nextX}(\text{OldB}, \text{NewB}):- \\
\quad \text{getXPosition}(\text{N}), \quad \% \text{ ask where to play} \\
\quad \text{checkPosition}(\text{OldB}, \text{N}),!, \quad \% \text{ is it available} \\
\quad \text{setPosBoard}(\text{OldB}, \text{N}, \text{x}, \text{NewB}). \\
\quad \% \text{ set the board to x}
\]

\[
\text{nextX}(\text{OldB}, \text{NewB}):- \quad \% \text{ else error message} \\
\quad \text{write('Not an empty board position!')}, \\
\quad \text{nl}, \\
\quad \text{nextX}(\text{OldB}, \text{NewB}).
\]
Board example:
Checking a position on board

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

```prolog
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

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

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

– 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 `search(X, LL)` which succeeds if X is a data in a linked list LL.

```
search(_, end):- fail.
search(X, llist(X, _)).
search(X, llist(_,Rest)) :- search(X, Rest).
```

• Write `insert(X, LL1, LL2)` which inserts X in front of LL1 to get LL2.

```
insert(X, LL1, llist(X, LL1)).
```

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

• Write \textit{add}(X, LL1, LL2) which inserts X in an \textbf{ordered} link list LL1 to get LL2.

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

?- add(34, end, R1), add(31, R1, R2),
    add(2, R2, R3), add(69, R3, Result).
R1 = llist(34, end),
R2 = llist(31, llist(34, end)),
R3 = llist(2, llist(31, llist(34, end))),
Result= llist(2, llist(31, llist(34, llist(69, 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:
  \[
  \text{del}(X, \text{Old}, \text{New}) :- \text{add}(X, \text{New}, \text{Old}).
  \]

  \[
  \text{?- add}(5, \text{llist}(1, \text{llist}(2, \text{end})), \text{R}), \\
  \text{del}(2, \text{R}, \text{Final}).
  \]
  \[
  \text{R} = \text{llist}(1, \text{llist}(2, \text{llist}(5, \text{end}))), \\
  \text{Final} = \text{llist}(1, \text{llist}(5, \text{end})); \text{false}.
  \]
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’.

  \[ t(t(t(\text{end}, \text{6}, \text{end}), \text{1}, t(t(\text{end}, \text{2}, \text{end})), \text{5}, t(\text{end}, \text{6}, \text{end}))) \]

![Binary Tree Diagram](image-url)
Counting the elements in a binary tree

\[
\text{\texttt{count}}(\text{end}, 0).
\]
\[
\text{\texttt{count}}(t(\text{Left}, \text{Root}, \text{Right}), N) :-
\quad \text{\texttt{count}}(\text{Left}, N1),
\quad \text{\texttt{count}}(\text{Right}, N2),
\quad N \text{ is } N1 + N2 + 1.
\]

?- \texttt{count}(t( t( t(\text{end}, 6,\text{end}),1,
\quad t(\text{end},2,\text{end})), 5, t(\text{end},6,\text{end})),
\quad N).
\]
\[
N = 5.
\]
Sorted Binary Trees

- A binary tree is sorted if $Left < Root \leq Right$

- Searching for an item in a sorted binary tree:

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

- Adding an item in a sorted binary tree

```prolog
addT(X, end, t(end, X, end)). % if empty tree
addT(X, t(L, Root, R), t(L1, Root, R)):-
    X < Root,
    addT(X, L, L1).
addT(X, t(L, Root, R), t(L, Root, R1)):-
    X >= Root,
    addT(X, R, R1).
```

?- addT(1, t(t(end,2,end),5,t(end,6,end)), L).
L = t(t( ? ,5,t(end,6,end)))
L = t(t( ? ,2,end),5,t(end,6,end)))
L = t(t( t(end,1,end),2,end),5,t(end,6,end)))
Sorted Binary Trees - add items

After addT(1, ..)
• 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 \( \text{delMin}(T1, Y, T2) \)?