STIN2024

Pengaturcaraan Logik
February
2\textsuperscript{nd} Session 2014/2015 (A142)

Wan Hussain Wan Ishak

School of Computing
UUM College of Arts and Sciences
Universiti Utara Malaysia

(P) 04-9285150
(E) hussain@uum.edu.my
(U) http://wanhussain.com
Lecture notes

List, Operators and Arithmetic

- Arithmetic
- Constructing Expressions
- List Processing
- Character Strings and Structures
List

- List is one of the most important Prolog data structures.

- A list is an ordered sequence of zero or more terms written between square brackets and separated by commas.

  \([a, b, c, d]\)
List
List

\[\{a, b, c, d\}\]

The elements of a list can be any kind of Prolog terms, including other lists.

\[\{1, 2, 3, 4\}\] \[\{a1, a2, a3\}\]
\[\{\text{sawi}, \text{kangkung}\}\] \[\{\text{kucing(comel)}, \text{kucing(hitam)}\}\]
\[\{\{\text{ satu}, \text{dua}\}, \{\text{tiga}, \text{empat}\}\}\]
List

- The empty list is written:

\[
[]
\]

- Please note one element list \([a]\) is not equivalent to the atom \(a\).

\(? - [a] = a.\)

no
List

- List can be constructed or decomposed through unification.

<table>
<thead>
<tr>
<th>Unify with</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
  [a, b, c] = X                                   | X = [a, b, c]              |
| 
  [X, b, Z] = [a, Y, c]                           | X = a, Y = b, Z = c        |
| 
  [[a, b], c] = [X, Y]                            | X = [a, b], Y = c          |
| 
  [a(b), c(X)] = [Z, c(a)]                        | X = a, Z = a(b)
List

- More examples

<table>
<thead>
<tr>
<th>Unify with</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>([X</td>
<td>Y]) = [a,b,c,d]\</td>
</tr>
<tr>
<td>([X</td>
<td>Y]) = [a]</td>
</tr>
<tr>
<td>([X, Y</td>
<td>Z]) = [a,b,c]\</td>
</tr>
<tr>
<td>([X, Y, Z</td>
<td>A]) = [a,b,c]\</td>
</tr>
<tr>
<td>([X, Y</td>
<td>Z]) = [a</td>
</tr>
</tbody>
</table>
List

- The list can be divided into head and tail by the symbol ‘|’.

\[
[H \mid T]
\]

- The first element is the head and the rest are the tail.

Example:

\[
[a \mid [b,c,d,e]]
\]
List

\[ a | [b, c, d, e] \]

- The tail of a list is always a list, the head of a list is an element.

- Every nonempty list has a head and a tail.

\[ [a, b, c, d] = [a \mid [b, c, d]] \]
\[ [a] = [a \mid []] \]
List Manipulation

- **Built-in predicates**

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>append/3</code></td>
<td>join or split lists</td>
</tr>
<tr>
<td><code>length/2</code></td>
<td>get the length of a Prolog list</td>
</tr>
<tr>
<td><code>member/2</code></td>
<td>get or check a member of a list</td>
</tr>
<tr>
<td><code>member/3</code></td>
<td>get or check a member of a list and its position</td>
</tr>
<tr>
<td><code>remove/3</code></td>
<td>remove an element from a list</td>
</tr>
<tr>
<td><code>removeall/3</code></td>
<td>remove all occurrences of an item from a list</td>
</tr>
<tr>
<td><code>reverse/2</code></td>
<td>check or get the reverse of a list</td>
</tr>
</tbody>
</table>
List Manipulation

append/3

Syntax:

append(First, Second, Whole)

Example:

Join list

?- append([a,b], [c,d], Whole).
    Whole = [a,b,c,d]

Splitting list

?- append([a,b], Second, [a,b,c,d]).
    Second = [c,d]

?- append(First, [c,d], [a,b,c,d]).
    First = [a,b]
List Manipulation

- length/2

Syntax:

```
length(Term, Length)
```

Example:

```
?- length([a,b,c,d], Length).
Length = 4
```
List Manipulation

- **member/2**

**Syntax:**

`member(Element, List)`

**Example:**

?- member(a, [a,b,c,d]).
yes

?- member(Element, [a,b,c,d]).
Element = a ;
Element = b ;
Element = c ;
Element = d ;
List Manipulation

- member/3

Syntax: \[\text{member(Element, List, Position)}\]

Example:

\[
\begin{align*}
\text{Element} & \quad \text{position} \\
? & \text{- member(c, [a,b,c,d], Position).} \\
& \text{Position = 3 ;} \\
\end{align*}
\]

\[
\begin{align*}
\text{Get Element} & \quad \text{at given position} \\
? & \text{- member(Element, [a,b,c,d], 2).} \\
& \text{Element = b} \\
\end{align*}
\]
List Manipulation

- member/3 (more example):

```
?- member(Element, [a,b,c,d], Position).
Element = a ,
Position = 1 ;

Element = b ,
Position = 2 ;

Element = c ,
Position = 3 ;

Element = d ,
Position = 4 ;

Get the element and its position
```
List Manipulation

- remove/3

Syntax:

\[
\text{remove} (\text{Element, List, Remainder})
\]

Example:

- Takeout an element
  - \(?- \text{remove}(b, [a,b,c,d], \text{Remainder}).
    \text{Remainder} = [a,c,d] ;
  - Element missing?
    - \(?- \text{remove}(\text{Element, [a,b,c,d], [a,b,d]}).\)
      \text{Element} = c ;

List Manipulation

- remove/3 (more example):

?- remove(Element, [a,b,c,d], Remainder).
Element = a,  
Remainder = [b,c,d];

Element = b,  
Remainder = [a,c,d];

Element = c,  
Remainder = [a,b,d];

Element = d,  
Remainder = [a,b,c];

What element can be removed
List Manipulation

- removeall/3

Syntax:

removeall(Item, List, Remainder)
List Manipulation

- removeall/3 (Example):

  **Remove all repeated elements**

  ```prolog
  ?- removeall(a, [a,b,a,b,a], Remainder).
  Remainder = [b,b]
  ```

  **Remove all elements that match with the first element**

  ```prolog
  ?- removeall(Item, [a,b,a,b,a], Remainder).
  Item = a,
  Remainder = [b,b]
  ```

  ```prolog
  ?- removeall(Item, [b,a,b,a], Remainder).
  Item = b,
  Remainder = [a,a]
  ```
List Manipulation

- Reverse/2

Syntax:

\[
\text{reverse}(\text{List, Revlist})
\]

Example:

\[
\text{?- reverse([a,b,c,d], Revlist).}
\]

Revlist = [d,c,b,a]

\[
\text{?- reverse(List, [d,c,b,a]).}
\]

List = [a,b,c,d] ;

Error 4, Heap Space Full, Trying ewrite/1
List Manipulation - Exercise

What is the output?

?- append([ab],[b,c,d], X).

?- reverse([b,c,d], R), append([ab], R, X).

?- reverse([b,c,d], R), member(F, R, 1), remove(F, R, B), append([ab], B, X).
List – Constructing Predicates

- `append/3`
- `length/2`
- `member/2`
- `member/3`
- `remove/3`
- `removeall/3`
- `reverse/2`

✓ How to define or construct the predicates?
✓ How they are working/How they manipulate the list?
List – Constructing Predicates

- **append/3** $\rightarrow$ **addon/3**

**Syntax:**

```
addon(First, Second, Whole)
```

**Definition:**

```
addon ([],X,X).  $\leftarrow$ stop the recursion

addon ([H|T1],X,[H|T2]):-  $\leftarrow$ add element into the list
    - addon (T1, X, T2).
```
List – Constructing Predicates

append/3  →  addon/3

?- addon([a,b],[c],W).

Ep. 1

Call (1)  →  addon([],X,X).

Call (2)  →  addon([H|T1],X,[H|T2]).

Call (2.1)  →  addon(T1,X,T2).

addon([b],[c],T2).  →  go to Ep. 2
List – Constructing Predicates

append/3 → addon/3

From Ep.1, (2.1) → addon([b],[c],T2).

Call (1) → addon([],X,X).

Call (2) → addon([H|T1],X,[H|T2]).

Call (2.1) → addon(T1,X,T2).

false \[b\]=[]

true \[b\]=[H|T1] \[c\]=X \[H\mid T2\]=W

H=b
T1=[]
X=[c]
W=[b|T2]
T2=_

addon([],[],T2). → go to Ep. 3
List – Constructing Predicates

append/3 → addon/3

From Ep.2, (2.1) → addon([],c,T2).

Ep. 3

Call (1) → addon([],X,X).

Stop at Ep. 3 rule (1). Rule (2) will not execute.
List – Constructing Predicates

?– addon([a,b],[c],W). W = [a,b,c]

Ep. 1 (2)  addon([a|b],[c],[a|T2]):– T2 = [b,c]

Ep. 2 (2)  addon([b|[]],[c],[b|T2]):– T2 = [c]

Ep. 3 (1)  addon([], [c], [c]).
List – Constructing Predicates

- `length/2` → `noOfTerms/2`

Syntax:

```prolog
noOfTerms(Term, Length)
```

Definition:

```prolog
noOfTerms([], 0).  % stop the recursion
noOfTerms([H | Tail], K) :-
    noOfTerms(Tail, J),
    K is J + 1.  % calculate the length
```
List – Constructing Predicates

\[ \text{length/2} \rightarrow \text{noOfTerms/2} \]

?- \text{noOfTerms([a,b,c],T)}. 

Ep. 1

Call (1) \rightarrow \text{noOfTerms([],0)}.

Call (2) \rightarrow \text{noOfTerms([H|Tail],K)}.

\rightarrow \text{Call (2.1) \rightarrow noOfTerms(Tail,J)}.

\rightarrow \text{noOfTerms([b,c],J).} \rightarrow \text{go to Ep. 2}

Call (2.2) \rightarrow \text{pending. (At the moment Sub goal 2.2 will not be called.)}

(1) \text{noOfTerms([],0)}.

(2) \text{noOfTerms([H|Tail],K):-}

(2.1) \text{noOfTerms(Tail,J),}

(2.2) \text{K is J + 1.}

\text{false} \quad [a,b,c] \neq []

\text{true} \quad [a,b,c]=[H|Tail]

H=a

T=K

Tail=[b,c]

T=K=_
List – Constructing Predicates

\[ \text{length/2} \rightarrow \text{noOfTerms/2} \]

From Ep.1, (2.1) \(\rightarrow\) noOfTerms([b,c],J).

Call (1) \(\rightarrow\) noOfTerms([],0).

Call (2) \(\rightarrow\) noOfTerms([H|Tail],K).

\(\Rightarrow\) Call (2.1) \(\rightarrow\) noOfTerms(Tail,J).

\(\Rightarrow\) Call (2.2) \(\rightarrow\) pending. (At the moment Sub goal 2.2 will not be called.)

Ep. 2

noOfTerms/2

\(1\) noOfTerms([],0).

\(2\) noOfTerms([H|Tail],K):-

(2.1) noOfTerms(Tail,J),

(2.2) K is J + 1.

false [b,c]=[]

true [b,c]=[H|Tail]

\[ J=K \]

H=b

Tail=[c]

J=K=\_

noOfTerms([c],J) \(\rightarrow\) go to Ep. 3
List – Constructing Predicates

\[ \text{length/2} \rightarrow \text{noOfTerms/2} \]

From Ep.2, (2.1) \( \rightarrow \) noOfTerms([c],J).

\[ \text{Ep. 3} \]

Call (1) \( \rightarrow \) noOfTerms([],0).

Call (2) \( \rightarrow \) noOfTerms([H|Tail],K).

\[ \text{Call (2.1)} \rightarrow \text{noOfTerms(Tail,J)}. \]

\[ \text{Call (2.2)} \rightarrow \text{pending. (At the moment Sub goal 2.2 will not be called.)} \]

\[ \text{false} \quad [c]\neq[] \]

\[ \text{true} \quad [c]=[H|Tail] \]

\[ \text{H=}c \]
\[ \text{Tail=}[] \]
\[ \text{J=}K \]
\[ \text{J=}K=_ \]

noOfTerms([],J). \( \rightarrow \) go to Ep. 4
List – Constructing Predicates

- length/2 → noOfTerms/2

From Ep.3, (2.1) → noOfTerms([],J).

Call (1) → noOfTerms([],0).

Ep. 4

true []=[] J=0

Prolog will now return to Ep. 3 to execute 2.2. Followed by Ep. 2 and Ep. 1.
List – Constructing Predicates

? - noOfTerms([a,b,c],T).

1

noOfTerms([],0).

2

noOfTerms([H|Tail],K):-
noOfTerms(Tail,J),
K is J + 1.

(2.1)

noOfTerms([a|[b,c]],K):-
noOfTerms([b,c],J),
K is J + 1.

(2.2)

noOfTerms([a|[b,c]],K):-
noOfTerms([b,c],J),
K is J + 1.

(2.2)

noOfTerms([b|[c]],K):-
noOfTerms([c],J),
K is J + 1.

(2.1)

noOfTerms([c|[ ]],K):-
noOfTerms([ ],J),
K is J + 1.

(2.2)

K = 0 + 1

T = 3

yes
List – Constructing Predicates

\[
\text{noOfTerms}([a|[b,c]],K).
\]

- \( K \) is 2 + 1
  - \( J = K = 2 \)
  - \( K \) is 1 + 1
    - \( J = K = 1 \)
    - \( K \) is 0 + 1
      - \( J = K = 0 \)
      - \( \text{noOfTerms}([],0). \)

- \( K \) is 1 + 1
  - \( \text{noOfTerms}([b|[c]],K) \)

- \( K \) is 0 + 1
  - \( \text{noOfTerms}([c|[]],K) \)

- \( K \) is 0 + 1
  - \( \text{noOfTerms}([],0). \)
List – Constructing Predicates

**member/2** $\rightarrow$ **is_in/2**

**Syntax:**

\[\text{is}\_\text{in} \ (\text{Element}, \ \text{List})\]

**Definition:**

\[
\begin{align*}
\text{is}\_\text{in} \ (X, [X|T]). & \quad \leftarrow \text{check if the Element is Head of the list} \\
\text{is}\_\text{in} \ (X, [H|T]):- & \quad \leftarrow \text{if not, traverse the rest of the list (Tail)} \\
\text{is}\_\text{in}(X, T). & \quad \end{align*}
\]
List – Constructing Predicates

\[ \text{is\_in}(X, [a, b, c]). \]

- \( X = a \)
  - \( \text{is\_in}(X, [b, c]). \)
    - \( X = b \)
      - \( \text{is\_in}(X, [c]). \)
        - \( X = c \)
          - Return \( X = c \)
    - Return \( X = b \)
  - Return \( X = a \)

- \( \text{is\_in}(X, [b, c]). \)
  - Return \( X = b \)

- \( \text{is\_in}(X, [c]). \)
  - Return \( X = c \)

- \( \text{is\_in}(X, []). \)
  - Return \( X = a \)
  - Return \( X = b \)
List – Constructing Predicates

\[
is_{\text{in}}(c, [a, b, c]).
\]

\[
c \neq a \quad \text{is}_{\text{in}}(c, [b, c]).
\]

\[
c \neq b \quad \text{is}_{\text{in}}(c, [c]).
\]

\[
c = c \quad \text{is}_{\text{in}}(c, []).
\]

Return yes
List – Constructing Predicates

- remove/3 $\rightarrow$ take_out/3

Syntax:
\[
\text{take}_\text{out} \ (\text{Element}, \text{List}, \text{Remainder})
\]

Definition:
\[
\text{take}_\text{out} \ (X, [X|R], R). \quad \leftarrow \text{element found}
\]
\[
\text{take}_\text{out} \ (X, [F|R], [F|S]):-\text{take}_\text{out} \ (X, R, S). \quad \leftarrow \text{traverse the list, at the same time create new list}
\]
List – Constructing Predicates

- \( \text{reverse}/2 \rightarrow \text{overturn}/2 \)

**Syntax:**
\[
\text{overturn} \ (\text{List}, \text{Revlist})
\]

**Definition:**
\[
\text{overturn} \ (\text{List},\text{RevList}):- \\
\text{overturn2} \ (\text{List},[],\text{RevList}).
\]

\[
\text{overturn2} \ ([H|T],Z,W):- \\
\text{overturn2}(T,[H|Z],W). \leftarrow \text{the element position}
\]

\[
\text{overturn2}([],X,X).
\]
Character Strings

Three ways to represent a string of characters in Prolog:

- As an atom – atoms are compact but hard to take apart or manipulate.
- As a list of ASCII codes – can use standard list processing techniques on them.
- As a list of one-character atoms - can use standard list processing techniques on them.
Character Strings

Use write/1 with double quotes.

Example:

?- write(“abc”).

[97,98,99]
Character Strings

- Use put/1 or putb/1.

Example:

```prolog
write_str([H|T]):-
    put(H), write_str(T).

write_str([]).
```
Character Strings

- Convert between an atom or number and a byte list - name/2.

- Syntax:
  
  name(Atomic, List)

Example:

?- name(makan,List).
List = [109,97,107,97,110]

?- name(Atomic, [109,97,107,97,110]).
Atomic = makan
Character Strings

Example (Prolog codes):

```prolog
splitNprint(Ayat):-
    name(Ayat,AyatList),
    write_str(AyatList).

write_str([H|T]):-
    put(H), write_str(T).

write_str([]).
```

Output:

```prolog
?- splitNprint(ahmad).
ahmad
```
Character Strings (Exercise)

Write a code/rule to print out as follows:

?- print_splits("prolog").
prolog
pro olog
pro log
prolo g
prolog
yes

print_splits(Word):-
    print_split(Word,Word).
print_split([],Word).
print_split([_|T],Word):-
    append(H,T,Word),
    write_str(H),
    write(' '),
    write_str(T),nl,
    print_split(T,Word).
write_str([]).
write_str([H|T]):-
    put(H),
    write_str(T).
Character Strings (Exercise)

- Write a code/rule to check whether the word end with “s” or not:

```prolog
?- end_with_s("flowers").
yes

?- end_with_s("car").
no

?- end_with_s("cars").
yes
```

```
end_with_s(List):-
    length(List, L),
    member(115, List, L).
```